(2)

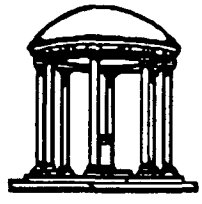
Automated Protocol Analysis: Tools and Methodology

TR91-034

August, 1991

John B. Smith
Dana Kay Smith
Eileen Kupstas

The University of North Carolina at Chapel Hill
Department of Computer Science
CB#3175, Sitterson Hall
Chapel Hill, NC 27599-3175
919-962-1792
jbs@cs.unc.edu



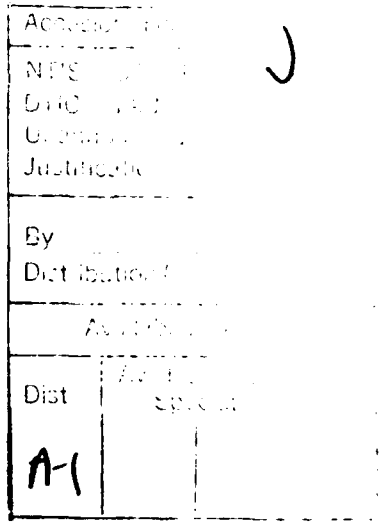
A TextLab/Collaboratory Report

Portions of this research were supported by the National Science Foundation (Grants # IRI-8519517 and # IRI-8817305), The Army Research Institute (Contract # MDA903-86-C-345), and The Office of Naval Research (Contract # N00014-86-K-00680).

Abstract

The TextLab Research Group, over the past seven years, has developed a collection of tools and techniques for recording users' interactions with computer systems in machine-readable form and for automatically analyzing and displaying those data. This report catalogs those tools and discusses their methodological context and implications.

Tools discussed include the following: *tracking* users' behaviors and producing a machine-recorded protocol at the level of action -- data an order of magnitude larger than keystrokes; *replaying* users' sessions from protocol data; modeling users' strategies using formal *cognitive grammars*; analyzing user sessions by *parsing* them with the grammars; and *displaying* results in visual form -- both static and animated -- to facilitate interpretation and understanding by researchers. These tools are placed in a methodological context by reviewing issues associated with concurrent think-aloud, keystroke, and video protocols; and other computer systems are reviewed that support these forms of protocol data. The discussion concludes by noting the increased importance of data management required by automated methods and our thoughts for a comprehensive environment, built around a protocol management facility, that would integrate the tools discussed and support development of new ones.



Introduction

The TextLab Research Group has developed a collection of tools and techniques for recording users' interactions with computer systems in machine-readable form and for automatically analyzing and displaying those data. The purpose of this report is to catalog those tools and to discuss their methodological context and implications.

We built these tools to study users' strategies as they work with systems developed by our group. A cornerstone of our research is the goal of amplifying users' intelligence for open-ended tasks, such as planning and writing technical documents or developing software systems. To build such *Intelligence amplification* (IA) systems, we must also build an understanding of users' conceptual strategies and conceptual processes for the task being supported. This requires techniques that can be applied to coherent behaviors, whose duration can extend as long as several hours, several weeks, or longer. Tools and methods must be able to infer broad, strategic patterns that cover the entire period while also identifying specific processes that last for only a few seconds. As we explain in more detail below, machine-recorded transcripts -- called *protocols* -- offer an attractive source of data for these purposes.

In the section that follows, we begin by reviewing methodological issues for traditional and machine-recorded protocols. Next, we briefly describe tools developed by other researchers to work with these different kinds of protocol data. After that, we provide an overview of our approach by tracing the flow of protocol data through the various tools we have developed.

Background

Since human thought processes cannot be observed directly, cognitive psychologists and others studying human intellectual behavior have developed several approaches for observing these processes indirectly. The methodology of controlled experiments has been finely honed for examining specific behaviors under laboratory conditions. For activities that involve combinations of mental skills or that can't be carried out in the laboratory, other methods are required. One such method, called *concurrent think-aloud protocols*, was developed and used by Newell, Simon, and others at Carnegie-Mellon University during the 1960s in order to study complex, problem solving behaviors [Newell & Simon, 1972]. An alternative method, called the *keystroke method*, has been used for tasks that involve computer systems; it is based on data comprised of each key pressed by a subject while working with a computer system [Card, Moran, & Newell, 1983]. A third approach, based on video recordings, has been used both as a supplement to think-aloud and keystroke methods and in its own right. Our group has developed a fourth approach that focuses on users' *actions*, recorded while they work with visual, direct manipulation systems. In the discussion that follows, we briefly outline issues raised by these different kinds of concurrent protocols.

Think-aloud protocols have provided a rich source of information for cognitive psychologists and for those studying human-computer interaction. The goal of this method is to produce a written record of subjects' trains of thought based on the subjects' own verbalization of their thinking as it occurs while they perform the task being investigated. Tasks that have been studied using this method include writing

documents, solving arithmetic problems, assembling physical devices, and playing chess. Recently, the technique has also been used to study human-computer interaction. Under laboratory conditions, subjects are prompted by the experimenter to continuously narrate their thoughts; however, under naturalistic conditions, such as a subject's writing a paper at home, such prompting is impractical. Consequently, think-aloud protocols often differ significantly in their level of detail.

Think-aloud protocols have been questioned on several other grounds, in addition to level of detail. Nisbett and Wilson [1977] raised three kinds of questions. First, subjects may have incomplete knowledge of their thinking processes; for example, experts frequently have difficulty explaining how they solve complex technical problems. Consequently, the protocol record may be incomplete in a more fundamental sense than amount of detail. Second, subjects may not have an accurate understanding of the processes they are aware of. Perhaps most serious of all, however, is the possibility that the act of thinking aloud, itself, may distort the subject's behavior: the way a subject carries out a task while thinking aloud may not be the same as when the subject does not think-aloud. This last objection calls into question the basic design of experiments that rely on think-aloud protocols.

Ericsson and Simon [1980; 1984] constructed a response to these concerns by reviewing a large volume of prior research, including their own studies. They argued that concurrent think-aloud protocols do, in fact, constitute valid data for *most* tasks. Their position is based upon distinctions they make among three types of tasks, which they call Level 1, Level 2, and Level 3 conditions. Level 1 tasks are those in which verbalization of concepts is an inherent part of the task and the verbalization is successively stored in short term memory in verbal form throughout the task. Under these conditions, they found no evidence that thinking-aloud affects this type of cognitive processing or that resulting protocol data are incomplete or distorted. Level 2 tasks are those in which verbalization may be part of the cognitive process but would not normally be heeded as part of that task. Level 3 tasks are those in which verbalization is not part of the cognitive process and, hence, must be generated for the think-aloud process, itself. For both Level 2 and Level 3 conditions, Ericsson and Simon concluded that think-aloud protocols *can* significantly change the cognitive process, especially for tasks that involve recognizing complex patterns and relationships presented visually and for abstract conceptualization [Ericsson & Simon, 1980; Ericsson & Simon, 1984; Claparède, 1934; Henry, 1934].

For researchers concerned with computer systems in which users represent abstract ideas visually and work with them through direct manipulation of associated icons, Level 2 and Level 3 conditions strongly apply. Thus, using think-aloud protocols to study these users' cognitive behaviors *should* be expected, from a theoretical perspective, to result in distortions of the data they produce, under at least some conditions. This conclusion does not argue definitively against their use in studying human-computer interaction, but it does suggest caution.

In addition to theoretical issues of validity, think-aloud protocols also present problems of interpretation. Typically, they are not used in raw form but, rather, are coded according to a taxonomy of categories [Swarts, Flower & Hayes, 1984]. The problem raised here is the consistency and accuracy among the human judges who code the raw protocols. While training can increase reliability and consistency among judges, encoding remains a subjective process that is prone to differences on the order of 25% [Hayes & Flower, 1980].

A third issue is practicality. An hour of think-aloud data typically produces fifteen to twenty pages of transcription [Hayes & Flower, 1980]. Producing this

transcript is a costly and time-consuming process that limits the number of subjects that can be studied and the range of questions that can be examined using this approach.

Recently, more subtle questions have been raised concerning the possible impact differences in verbal fluency among subjects may have on their think-aloud protocols. In an earlier study, Flower and Hayes [1984] found that one of the most important differences between the think-aloud protocols of expert writers and those for novice writers was the number of planning and structural revision actions subjects reported. Hayes [1989] has since posed the question of whether or not higher verbal fluency among experts may be responsible for this difference in the number of relevant statements made by the two groups as opposed to more attention actually being given by experts to planning and structural revision.

Thus, thinking-aloud is a technique that must be used selectively for theoretical reasons, while the costs and effort involved may require, for many studies, that it be used conservatively. Nevertheless, concurrent think-aloud protocols offer a source of rich and finely nuanced data that is unavailable by other means. Below, we suggest an approach, in which think-aloud and machine-recorded methods are used in conjunction with one another, that can alleviate many of the problems reviewed above.

For studies that involve computers, an alternative form of protocol data is sometimes available in the form of a record of each keystroke pressed by a subject. This approach is attractive for several reasons. Protocols may be recorded passively, unlike think-aloud methods. They can often be recorded by the operating system or by an analytic shell without modifying the application program. However, keystroke data also present several problems. First, they are very fine-grained; thus, storing, managing, and interpreting these data are formidable tasks. Second, to infer what the keystrokes add up to in terms of the commands and functions supported by the application program requires an analytic program equivalent to the application programs own internal user interface parser that interprets user commands. If interpreting a user's action is dependent on the current state of data in the application, the interpretive program would also have to duplicate much of the application program's function and data structures. A third problem concerns completeness: for systems that use graphic displays and mouse control, many user actions will not be recorded as keystrokes. Systems differ widely in their capabilities to record direct manipulation events. Consequently, for some systems, keystroke data will miss many, perhaps the majority, of the user's actions. Thus, while keystroke data solve the problem of distorting task performance posed by think-aloud protocols, they share the problems of incompleteness and producing large volumes of fine-grained data that must be analyzed or coded before being used.

A third method that has been used alone as well as in conjunction with both think-aloud and keystroke protocols is video recording. Subjects are video taped as they perform a task, perhaps while using a computer. These data can show what a person is doing when not thinking-aloud or not typing on the computer keyboard. They can also show what is being displayed on the computer screen -- information that is not likely to be available through the keystroke record. While video protocols provide a rich, new source of data, they also require extensive analysis and coding, raising the same issues of consistency and costs discussed above.

The UNC TextLab research group has developed an alternative approach that addresses many of these issues, but is restricted to studies that involve users of computer systems. Our approach is to record protocols at the level of users' *actions*, rather than keystrokes. Since we study users' strategies for systems developed by our project, we can embed sensors directly into our programs to record data, such as movements of the mouse from one window on the screen to another, selection of objects

or menu options, character strings typed in as names or labels, etc. Thus, the emphasis in these data is on users' interactions with visual objects through direct manipulation

These *action-level protocols* solve many of the problems discussed above for other forms of protocol data. They capture data frequently missed by keystroke protocols. They solve the problem of cognitive interference raised by think-aloud protocols by being passive and unobtrusive. And they address practical problems associated with volume of data by recording an order of magnitude fewer events than keystrokes. Like keystroke data, action-level protocols are recorded in machine-readable form, thus eliminating the need for manual transcription. Problems of analysis and interpretation remain, but they are simplified since atomic units at the action level are produced by sequences of keystrokes or events that have already been interpreted by the application program's own command processor.

Related Research

Several different kinds of computer tools have been developed to assist researchers working with the different kinds of protocols reviewed above. Here, we provide a brief overview of important examples of that work.

Think-aloud Protocols. Tools to assist analysts working with think-aloud protocols appeared almost as soon as that form of data; as might be expected, they were developed, in part, by Allen Newell whose pioneering work with Simon in human problem-solving introduced this new methodology [Newell & Simon, 1973]. The first such system, PAS-I, provided a set of powerful tools that included natural language parsing of think-aloud protocols and generation of graphs depicting sequences of subjects' mental actions [Waterman & Newell, 1971]. Whereas PAS-I was limited to working only with cryptarithmetic protocols, PAS-II was a more general tool that allowed input from the researcher during the parse and interpretation of the data [Waterman & Newell, 1973]. The trend toward generality has continued until, today, the predominant form of tool is one that manages the coding and analyses for the researcher, but assumes input from the human user for most, if not all, semantic information. These systems include VPA [Lueke, et. al., 1987], PAW [Fisher, 1988], and SHAPA [Sanderson, et.al., 1989]. VPA, a system developed within IBM, is intended primarily for testing system usability; it permits coding of protocols in terms of a hierarchy of categories and keywords defined by the user. PAW is intended primarily for research in programming skills. SHAPA is a general purpose tool that facilitates defining a structure for the protocol encoding vocabulary in terms of predicates and arguments, coding the protocols in terms of that structure, and collecting and aggregating data.

Video Protocols. A variety of tools have been developed for managing coded references to video protocols. Some of those described above can work with this form of data as well as think-aloud protocols; others designed for analyzing video protocols for groups will be discussed, below. Here, we focus on tools for working directly with the video medium, itself. To the best of our knowledge, this class of tool is still in the exploratory or development stages, but several examples reported in the literature illustrate the concept. Mackay [1989] discusses both the Experimental Video Annotator (EVA) she has developed as well as some of the issues it raises. Built as a prototype requiring an Athena workstation and special video hardware, EVA allows the user to view a video protocol and record time-stamps at points of interest as well as mark segments using previously-established codes. A text editor also allows the user to record comments linked to time-stamped sequences. Randy Trigg [1989] has described a similar tool, but with the addition of hypertext links and anchors that permit researchers to associate and to view in close succession different sequences on multiple

video tapes. Trigg also describes his plans for a more ambitious support system being planned at Xerox PARC.

Machine-Recorded Protocols. In their original description of the GOMS model, Card, Moran, and Newell [1983] describe the use of time-stamped records of keystrokes supplemented by hand-coded actions -- derived from either think-aloud or video protocols -- as a basis for modeling users' cognitive behaviors. A number of studies based on the GOMS and Keystroke models have followed; however, the basic tools and methodology have not been extended significantly since they were first introduced. Several researchers have used formal grammars to describe users' possible interactions with systems [e.g., Foley & Wallace, 1974; Reisner, 1981; Kieras & Polson, 1985], but they have not taken the next step and built parsing programs based on the grammars to analyze machine-recorded keystroke protocols. Our work in protocol tools and methods best fits within this category. We describe the tools and techniques we have developed, below; previously published discussions that may be of interest include [Smith & Lansman, 1989; Smith, Rooks, & Ferguson, 1989; Walker, 1991].

Group Protocols. Several projects have developed tools for studying groups from the points of view of cooperative work and/or collaboration. Most of these tools are currently oriented toward managing, analyzing, and displaying results based on data that is hand-coded from concurrent observations by a trained researcher or from subsequent analyses of video recordings. One such system is GroupAnalyzer, developed at the Capture Lab in the Center for Machine Intelligence in Ann Arbor, Michigan [Losada, et.al., 1990]. Using this system, trained observers code the behavior of each individual participating in a meeting in accord with a pre-established classification scheme, such as SYMLOG. GroupAnalyzer manages the coded data, including time-stamps for individual behaviors, and provides an interface for external programs that perform time-series and other analyses on the data. A similar tool, but with more extensive capabilities for displaying behaviors with respect to *artifacts*, is described in [Olson & Olson, 1991]. As more computer systems are developed to support cooperative and collaborative work, additional tools of this kind will be needed for working with machine-recorded group protocols. We describe some of our preliminary efforts in this direction, below, but at present, this work is still exploratory.

Overview

In this section, we provide an overview of the tools we have developed for working with action-level protocols by tracing the flow of data through them, as shown in Figure 1. Each tool or perspective introduced here is described in more detail, below.

Tracking. So that we may observe the actions performed by a user of one of our systems, we embed sensors in the program that note objects, and their positions, selected by the user with the mouse; menu options; time; and other relevant information. These data are formatted in accord with the rules of a *Protocol Description Language* that is general across all of our tools and then output to secondary storage.

Replay. Replay is the inverse function of tracking. We have modified our systems so that they may read-in a stored protocol from an earlier user session and recreate an approximation of the original session. Options permit the researcher to view the session in the same amount of time as the original, in proportional time, or in as brief a time as the system can perform the sequence of operations. Foreshortening time while viewing sessions helps the researcher to infer patterns and strategies, while the proportional time feature helps in getting a comparative sense of the amount of time required for different tasks and operations.

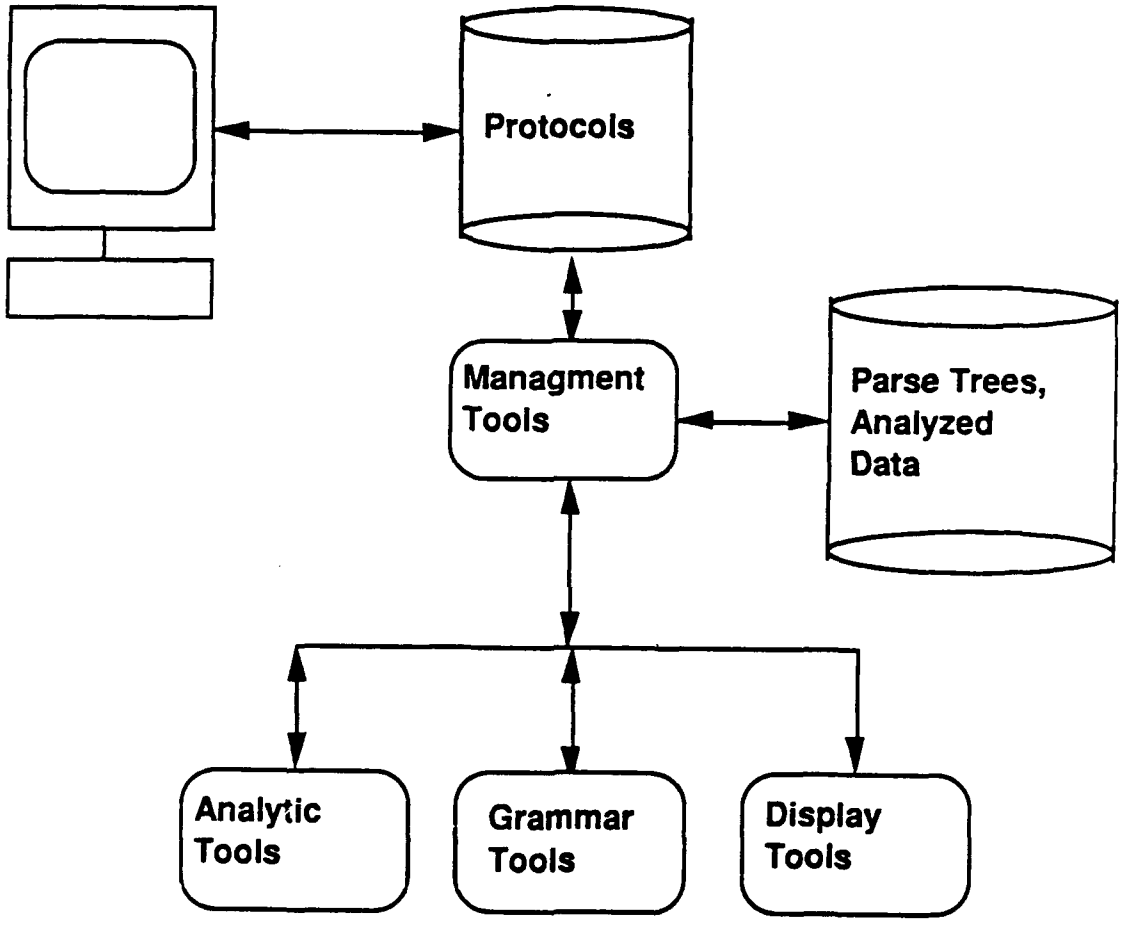


Figure 1:
Protocols Data Flow

Grammar Tools. Grammars are at the core of our theoretical and methodological thinking. On the theoretical side, grammars constitute formal models that describe users' cognitive interactions with computer systems for a particular conceptual task. On the methodological side, grammars are used by parsing programs to analyze machine-recorded protocols. This capability to automate analysis makes possible longitudinal and naturalistic studies for large numbers of subjects that would not be possible using conventional methods.

Analytic Tools. Both raw and parsed protocol data can be filtered to extract numeric values, time parameters, and other data that, in turn, are passed to external analysis programs.

Display Tools. While data may be analyzed by numerous automated tools, they must eventually be studied and interpreted by human beings who decide what they mean. To assist researchers with this essential task, we are developing an open-ended collection of visualization tools. We differentiate between *static* tools and *animated* tools. The first produce a fixed image of one or more protocols, or associated data, from a specific analytic point of view. The second is set of dynamic displays, usually shown on multiple workstation screens, coordinated by a replay of a session; as the session unfolds in time, the various visual displays showing the parsed or analyzed protocol update their data accordingly.

Data Management. Automated tools make it possible to gather and analyze large numbers of protocols. However, this richness of opportunity can be obscured by the volumes of data produced. Consequently, better data management tools are needed if the full potential of this methodology is to be realized. We are currently designing a comprehensive computer environment from which all protocol tools and data can be controlled. From it, we will be able to turn on and off the tracking function in our application systems; store and move protocol data; sort and select protocols from the database for analysis; develop, test, and modify grammars; parse specific protocols; apply various filter programs to the parse trees or raw protocol data to extract values that are then passed to statistical or other analytic programs; and, finally, control the various static and animated display programs.

Below, we discuss each of these tools in more detail.

Tracking

To produce a machine-recorded protocol of users interactions with a computer system, we embed sensors in the source code of our computer systems. They produce a data record for each action performed by a user, such as mouse movements, objects selected, and keyboard entries, along with the time (in milliseconds) of the action and other relevant attributes, such as the location of a node affected by the action. These data are formatted and written to external storage by the embedded tracker.

Data are recorded at the level of *event* or *action*, such as a menu selections or a sequence of several words entered as a parameter, such as the title for a node. Thus, the granularity of a protocol record based on actions is an order of magnitude larger than one based on keystrokes, such as those used with the GOMS keystroke model [Card, Moran, and Newell, 1983]. The reason for this difference in the granularity of protocol observations is that GOMS research has typically focused on fine-grained analysis of user interactions in order to predict performance times and particular sequences of low level commands selected under different conditions. The intent of our research is different; we are trying to understand broad strategies and patterns of behavior that

extend over hours for complex, open-ended tasks, such as planning and writing. User actions appear to be the lowest level of detail that can be said to represent acts of conceptual intent, whereas keystrokes or simple mouse movements are reflexive and, thus, lie below conceptual intent.

The recorded events are written out to secondary storage by the tracking module embedded in the application program. These data constitute what we call an *action level transcript* and serve as input to the session replay module and the protocol analysis tools described below.

The formats and semantics for our protocol transcripts have changed several times over the past six years. While it would seem straight-forward to report which event has taken place, at what time, lasting for what duration, at what point on the screen, etc., these simple requirements hide several subtle problems. We have managed to find at least some of them during the course of our work. Following is a short, history of the three different protocol formats we have used, followed by a discussion of the problems they have raised or addressed. We refer to these different formats as *versions 1-3*.

For all versions, the protocol file begins with a header containing the following information:

- o version of the Writing Environment used
- o date the transcript was created,
- o system time at which the session began
- o user name
- o clock time at the beginning of the session
- o name of the database for the session
- o boolean value indicating whether or not the database was empty at the beginning of the session

If the particular grammar allows different lexicons of protocol symbols to be used, another line is added to the header to indicate which lexicon is in use.

The header is followed by a blank line, then the sequence of actions which make up the session. The first event of a session is always *openSession*, with the mouse position at opening. The last event is always *closeSession*, with the mouse position at closing. The rest of the events indicate the behavior of the user.

Version 1 of the transcript language recorded:

- o elapsed time between events
- o event name
- o list of the attributes for the event.

The list of attributes is different for different action symbols; thus, it consists of zero or more attributes which may include identification of the object affected by the action, the position of the object on the screen, or the logical relationship of the affected object with respect to other objects in the environment. The events recorded in version 1 were

restricted to only those generated by user actions; in later versions, the tracker added information on its own to make the replay program more robust.

Version 2 of the transcript language recorded:

- o begin time of the event (relative to the beginning of the session)
- o event name
- o attributes.

This version includes several additional types of events that characterize user behaviors but are not associated with selecting a particular object. These events were added to the transcript language by the tracker to provide more detail needed by the replay program, such as events describing the user's roaming within the work space.

Version 3 of the transcript language records:

- o begin-time
- o empty field for the duration
- o event name
- o attributes

The duration of the event is calculated after the protocol has been parsed by a grammar. New events were added to aid replay and represent several previously unrecorded user actions.

Finding the right way to calculate the time attribute(s) for user actions has been difficult; indeed, as this brief history shows, we have used three different formulas. The main problem has been determining the duration of an event. In the early versions of our protocol language, we made the naive assumption that the duration of an event was the amount of time extending from the beginning of one event to the beginning of the next event. This is not accurate; the user often performs mental actions between events that don't result in computer actions -- such as planning, pondering, or day-dreaming. *Pauses*, thus, represent important information with respect to users' strategies and patterns of behavior that should be recorded for analysis.

We are currently implementing *version 4* of the transcript language; for each user action, it will record:

- o begin-time (relative to the session clock)
- o end-time (relative to the session clock)
- o event name
- o attributes.

Thus, the inactive time -- or pauses -- between events can be calculated and evaluated later during analysis.

Another mistake we will correct in version 4 concerns the format for the protocol file header. Earlier formats did not allow new information to be added to the header, since all down-stream tools assumed a fixed number of lines in the file header. Experience has shown that storing various information about particular experimental

conditions, the subject, the session, etc., to be useful during later analysis. Consequently, the header will be variable in length in version 4.

We are currently extending our work from addressing single users writing documents to groups of users developing software systems; thus, we need to develop tracking functions and protocol transcription formats that will permit us to study the ways in which a number of users interact with one-another in developing large, complex structures of ideas over periods of months to years. Our current level of protocol granularity may be appropriate for analyzing particular periods of work, but it may be impractical for extended studies because of the volume of data produced. A larger-grain record, that records the sequence of modes engaged by one or more users in order to read or work on a particular artifact, may be more practical, and it may provide more useful data for understanding collaborative interaction. An intermediate approach might be to record mode/artifact data for an entire collaborative project but record action-level protocols only for selected periods of work. Thus, we are considering making the granularity of the protocol events that will be recorded by our future systems a variable that can be changed dynamically.

Replay

So that we may observe the behaviors of users working with our systems, we have built facilities into them that can take an action-level protocol transcript, as described above, and recreate the user's session. We call this facility *replay*. Here, we describe replay as a stand-alone tool; below, we discuss its use in conjunction with other display tools.

During replay, the screen appears exactly as it appeared to the user during the session in which the protocol was recorded; however, none of the program's controls is active. Events are replicated from the beginning of the session, so the viewer can watch the user's strategy unfold. Along the bottom of the replay screen is a box showing the current protocol event and its time relative to the beginning of the original session. A sample replay screen is shown in Figure 2, which also includes a dynamic display tool in which the particular event is shown in context with other events classified as either *planning* or *writing* events. This tool will be presented in more detail later in our discussion.

Replay can be interrupted by the researcher at any point and, then, resumed. The viewer can also control the replay speed in several ways. The interval between events can be set so that it duplicates the pauses and the pace of the original session. Alternatively, it can be set to some proportional value of the original, such as one-tenth; this option permits the observer to view the session in a fraction of the original time but also gain a sense of the users "rhythm" of work. A third option lets the observer indicate a constant time interval that will be inserted between events; this interval can range from 0, in which case the session will replay as fast as the system can reproduce the user's actions, to any specified number of seconds. For an interval value of 0, a session that originally took the user two hours will replay in approximately 8-10 minutes. Finally, the observer may interrupt the replay and then manually step through events, one at a time or a specified number of events at a time. In this stepping mode of replay, the specified number of events are recreated, then the system pauses until the observer clicks the mouse, then the next set of events is shown, etc. Thus, replay allows the researcher to view a session in time proportional to the original, in uniform time, and in manually controlled steps.

Compressing the time required to recreate a user's session turns out to be surprisingly useful as a tool for analysis. Foreshortening session time allows one to *see*

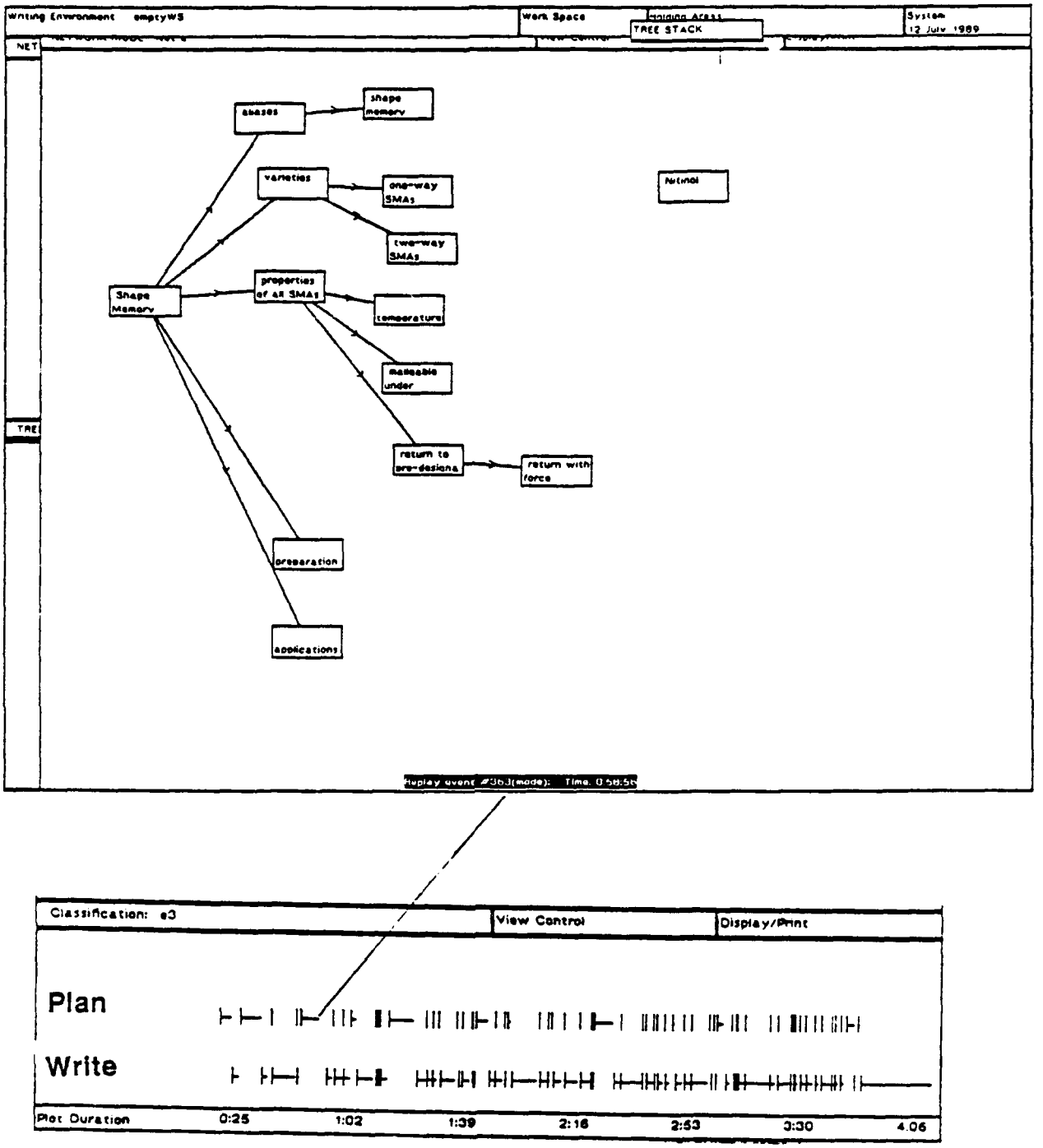


Figure 2:

Replay Screen

With Abbreviated Events-Time Tool

a user's strategy unfold during a session. For example, one can see the user layout a collection of nodes, note the order in which ideas were generated, see clusters transformed into more complex structures, note intervals of writing, per se, versus planning and editing, etc. One can also see the order in which different system modes were engaged and in which modes different ideas and structures were developed. At present, replay does not reproduce work carried out within the text editor; as a result, time spent in either of the editing modes is indicated by a blinking cursor for the appropriate duration, but no actual events are shown. In the future, we hope to extend our tracking and replay capabilities so that we may show the user's actions during actual writing.

Recently, we have begun using replay to gather cued retrospective think-aloud protocols as additional data for studies and to validate our grammars, described below. This form of think-aloud protocol is collected immediately after a working session is completed. The user and researcher view a replay of the session together, and the user narrates his or her actions, cued by the replay, for the entire session or for portions of particular interest to the researcher. In this way, the researcher can ask the subject about particular events or episodes without interrupting the original train of thought. This approach avoids the potential problem of task distortion associated with concurrent think-aloud methods [e.g., Nisbett and Wilson, 1977, Ericsson and Simon, 1980] while providing many of the benefits. In a future experiment, we will catalog more closely the similarities and differences in the data generated by the two methods as well as try to identify effects on the user's strategy of concurrent narration.

Replay seems like a simple function, but, as with protocol data formats, it raises subtle problems. Currently, the state of the replay is maintained by a finite state machine (FSM) embedded in the application system. The FSM knows the number of objects that must be updated for each type of event; however, for complex reasons, that count can occasionally become incorrect for some user actions, causing the FSM to wait indefinitely. Currently, the problem can only be circumvented by hand-editing the protocol transcript to remove the offending event; this is not a satisfactory solution. Other problems, not quite so fundamental, also exist. Attribute values that record the location of an event on the screen are recorded as absolute screen co-ordinates, which can cause problems when the replay is run on hardware different from the original -- for example, switching from Sun workstations to DEC workstations, which have a smaller screen, can cause replay to hang. Similarly, changing the application program source code can cause previously recorded transcripts to hang when we attempt to replay them on the new system. Problems can arise, for example, if application menu options are added or deleted, in which case old transcripts will not replay properly.

For future systems, we are considering recording two forms of protocols for each session. One protocol will be used strictly for replay. It will be a record of the low-level event queue produced by the system, from which the session can be recreated precisely. A second protocol will be recorded at higher levels of the system after low-level events have been interpreted in terms of their effects on the data objects being manipulated by the user. This second record, similar to our current action-level transcript, will be used by down-stream tools for analysis and for displays other than replay

Grammars and Related Tools

Concept

We view a protocol transcript as analogous to a statement or discourse in natural language. From this perspective, the symbols that represent individual user actions can

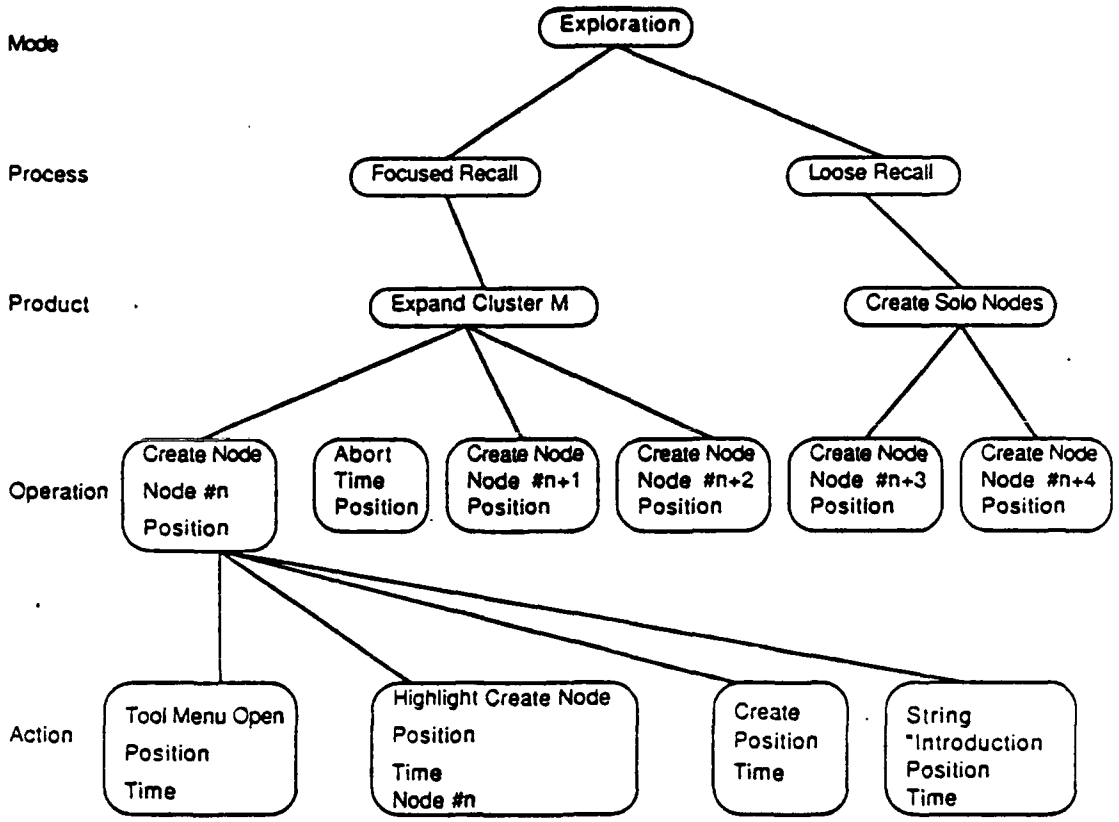
be thought of as words, and sequences of actions as cognitive phrases, sentences, or discourse, depicting the user's interactions with the system through which he or she attempts to achieve a hierarchy of goals. To analyze these data, we have developed formal cognitive models that are expressed as *grammars*. We use these grammars to *parse* sequences of protocol symbols -- analogous to parsing sequences of words using a natural language grammar. The resulting parse trees provide concrete representations of the user's cognitive behavior, tactics, and strategies, enabling us to make comparisons between sessions for different users or for the same user under different conditions

Our first grammar was expressed within the formalism of production rules supplemented by functions that recognize different types of graph objects, such as disconnected sets of nodes, connected graphs, trees, etc. That system was implemented using the OPS83 expert system shell. We are nearing completion of a second grammar expressed as an Augmented Transition Network (ATN), and we are currently working on an associated general-purpose parser. To assist with the refinement of this ATN model and with development of future ATN grammars, we are also building a visual, direct-manipulation grammar development tool. In the discussion that follows, we describe both grammars and the grammar development tool, as well as provide a brief background description of the ATN formalism.

Both grammars model the task of expository writing, including planning, writing, and revising activities, but with emphasis on the more structural, as opposed to linguistic, aspects of the process. Both grammars assume an underlying cognitive architecture for the kinds of thinking required for this and other similar tasks concerned with developing abstract structures of ideas (e.g., plans, software designs and implementations, military doctrine, etc.) The key assumption is that human beings engage a succession of different *cognitive modes* in order to carry out a particular task in accord with *tactics* and *strategies* that they have learned or developed. A mode is engaged in order to achieve a particular *goal* which is usually manifest as the production of some information artifact or *product*, such as an arrangement of notes, a plan for a document, or a paragraph of text. To produce this product within a particular cognitive mode, people use particular sets of cognitive *processes* and, by implication, do not use other processes. Behavior within a mode is further characterized by sets of *constraints* or rules specific for a given mode. Thus, a mode can be summarized as a way of thinking engaged in order to achieve a goal, resulting in the production of a particular kind of conceptual product, as a result of using a set of cognitive processes, in accord with a set of constraints and rules. For example, the task of expository writing often includes a form of early *exploratory* thinking during which writers brainstorm their topic, collect and analyze information, consider alternative perspectives, etc.; this kind of thinking is different form *organizational* thinking in which is developed that will guide *writing*, itself. These three modes of thinking are different from the several kinds of thinking used for revision, i.e., *structural, coherence,* and *linguistic editing*. For further discussion of cognitive modes, see [Smith & Lansman, 1989].

Production Rule Grammar

Our first grammar viewed a session as being composed of a series of *modes* in which cognitive *processes* were used to produce instances of particular *products* or changes to products. These conceptual products were represented and manipulated in the computer system by the user through various *operations*, each consisting of several specific system *actions*. A schematic representation of this model is shown in Figure 3. In this depiction, which stops at the level of mode rather than session, we see the four actions required to produce the node labelled *n*. These actions constitute the operation: *create_node*. In conjunction with several other operations, an existing *cluster* (a



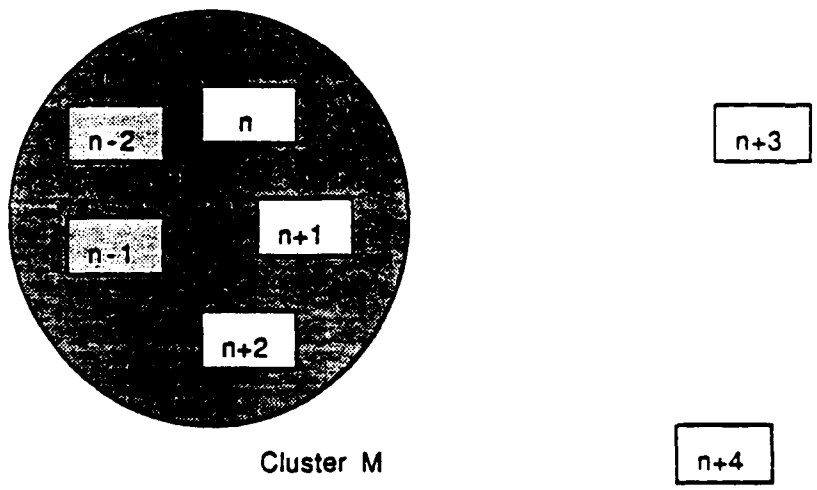


Figure 3:
Parse Tree Example

particular type of product) is being expanded through the process of *focused recall.* All of this is taking place within an *exploratory* mode of thinking.

The parser is actually a group of several different parsers that operate in a bottom-up manner in which the output produced at one level serves as the input to the next higher level. The data flow of these component parsers is shown in Figure 4. The action-level protocol transcript, produced and formatted by the tracker, is parsed to produce the operation-level protocol. The main parsing program receives this data as input and outputs, in parallel, the top three levels of the parse: the *delta-product-level,* the *process-level,* and the *mode-level* protocol transcripts. Thus, the parser produces horizontal slices of the parse three that are assembled to produce the parse tree for a session.

Next, we describe the implementation of these parsing programs. Before parsing begins, the action level transcript is filtered to remove events generated by the tracker, rather than the user, to facilitate replay. The action level to operation level analysis is then done by a yacc-based parser, since context-free rules are sufficient to recognize operations. Yacc provides facilities for defining the tokens (or symbols) in the grammar, the rules, and the output produced by each rule; it then converts these specifications into tables for a generalized LR(1) parsing algorithm. The tables are then used to parse the action level transcript, producing the operation level protocol transcript.

The three higher level protocol transcripts are produced by the expert system parser, written in OPS83 (declarative), plus additional context sensitive functions written in OPS83 (procedural), under the control of the main program. As the operation-level protocol symbols are read, the user's data structures are recreated. A set of *recognizer* functions then identify the effect of each operation on the data structure in terms of their effects on a small set of types of (intermediate) cognitive products, that include *isolated nodes, clusters* of nodes, *relations, trees,* and nontree *structures.*

The structures developed on the screen by the user and now recreated by the parser are assumed to represent the cognitive products and conceptual structures the user was trying to construct. The delta-product level analysis uses the recognizers and approximately forty production rules to make decisions about these conceptual products and the changes the user made to them during the session. The cognitive process level uses the changes in the cognitive products, as represented in the delta-product transcript, to make inferences about the cognitive processes that were engaged by the user, such as recalling ideas from memory, associating them, or expressing them as sentences. For example, creating several nodes in close spatial proximity (clustering) is interpreted as *focused recall,* whereas creating nodes that are spatially separate from one-another (unclustered) is considered *loose recall.* Approximately 21 rules are needed to identify the set of cognitive processes. The highest level protocol transcript is the cognitive mode level. Currently, the grammar recognizes four different modes: *explore, organize, edit,* and *revise,* where *edit* refers ambiguously to both *writing,* per se, and *linguistic editing.* The parser generates and outputs these three protocol transcripts -- delta product, process, and mode -- in parallel.

ATN Grammar

The cognitive grammar expressed in OPS83 works satisfactorily, but the architecture of the parsing system is awkward, making maintenance difficult. Our ideas concerning the model of writing have also evolved. Consequently, we began work this past year on a second grammar that will define a revised model of the writing process. That model is being developed in terms of the ATN formalism.

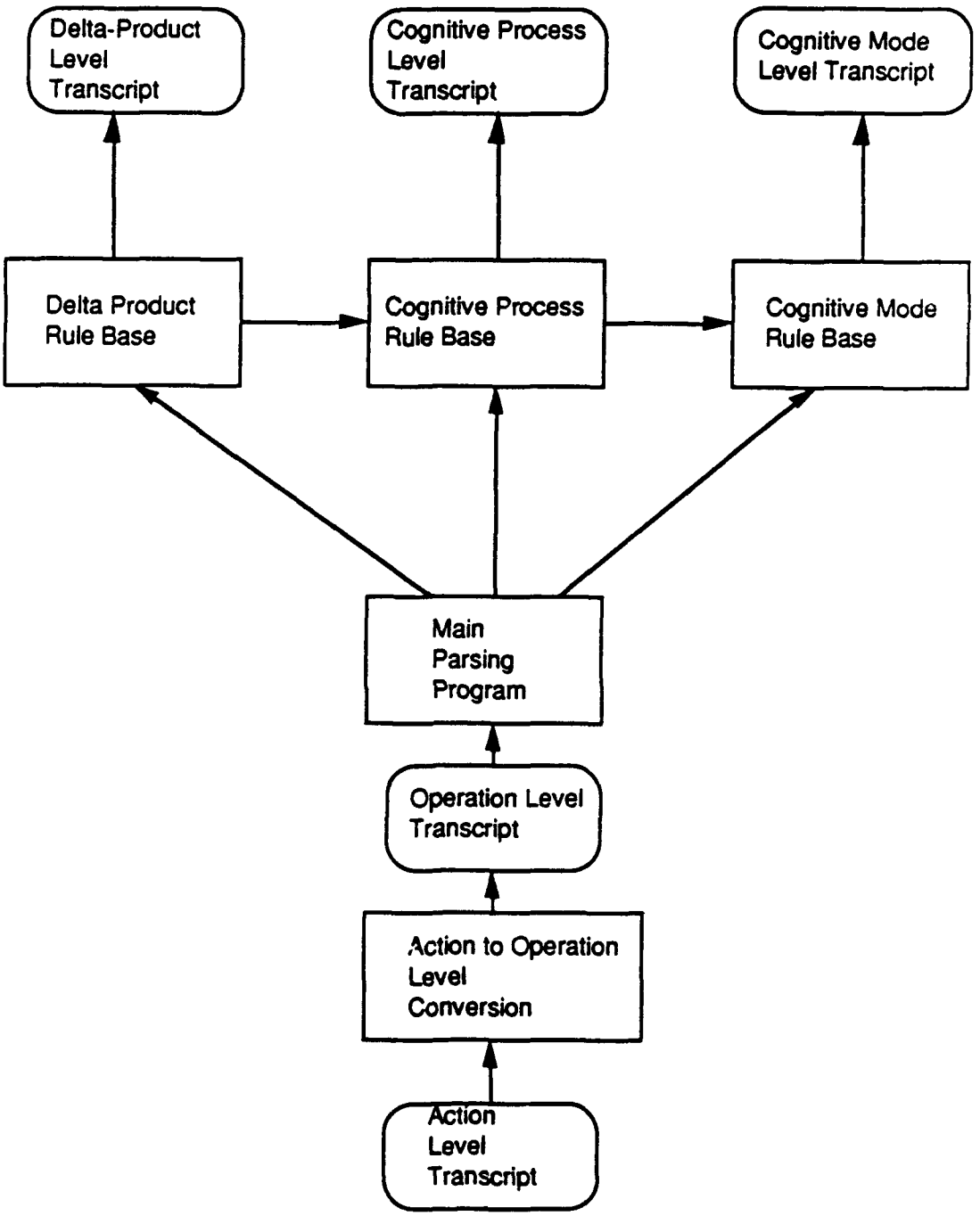


Figure 4:
Flow of Control for Expert-System-Based Grammar

An ATN is a specialized form of finite state machine (FSM) that allows *tests* on transitions, *operations* on *register* values, and *recursive* calls to other ATNs. ATNs were first described in [Woods, 1970]. The main application of ATNs has been in natural language understanding systems. However, the ATN formalism is quite powerful -- equivalent to a Turing machine -- and its register and test capabilities make it convenient for incorporating tests on parameter values -- such as the spatial coordinates of a node -- into the rules of the grammar.

An ATN grammar is expressed as a set of graph structures whose nodes represent states and whose links represent transitions. Descriptive labels that denote the nonterminal symbols of the grammar are normally attached to the links/transitions. A parse is performed by a program that reads symbols from a language and then traverses the ATN graph structures, checking for conditions, performing various tests, and recording information in various registers. The parse is complete when all symbols have been read in and the parsing program reaches a stop state; the parse tree for the string is constructed from the sequence of successful transitions in the path through the ATN graphs that ended in the stop state.

ATN graphs are traversed by moving from state to state over directed transitions. A transition may have several kinds of tests on it. An empty test allows unconditional crossing of the transition. Tests may check the category or the specific value of the current input symbol. For natural language, this test might determine whether the current word is a verb or a noun. Some ATN tests, such as *push* or *jump*, control movement of the parser between different ATN graphs. A push transfers control to another ATN named on the transition link, which can be a recursive call to itself. The named ATN must then be successfully traversed before the transition generating the push is completed. This structure allows the model represented by the ATN to be divided into layers. For example, an ATN representing sentence structure could have a push to a second ATN that recognizes noun phrases and a subsequent push to a third ATN that recognizes verb phrases. Other kinds of tests -- for example, in our application, computing the distance between two nodes on the screen -- may be defined on the links depending upon the specific use of the ATN.

A side-effect of making a transition from one state to another may be to set, send, or receive register values. Multiple registers can be defined, somewhat like local and global variables, to store specific data. In natural language applications, registers hold values for the subject of a sentence, the verb, etc. This information can be used by other parts of the ATN to check subject-verb agreement, gender agreement between the subject and a possessive pronoun, etc. Often, each part of the sentence is stored in a register so that at the end of the process, the entire sentence can be reconstructed along with the parse tree for that sentence.

Our decision to use the ATN formalism for our second grammar was motivated by several factors. Since ATNs have the power of a Turing machine, they can handle context -sensitive rules. Second, our group has worked with graph structures for so long, we tend to think in terms of that data model; since ATNs are graph structures, they are a natural formalism for our group to use. We can also reuse our hypermedia graph browsers as a basis for the specialized tools we are building to work with ATN networks. Third, interpreting users' cognitive strategies often involves testing attribute values, such as determining the distance between two nodes or the time between two events. Since ATNs include rules on arcs, they provide a convenient mechanism for handling this requirement. Thus, ATNs offer an attractive formalism for our particular project and our particular group.

The model of writing currently defined by our ATN grammar differs from our production rule model in several respects. Most obvious is that it includes a varying

number of levels. In the earlier grammar, all sequences of actions were ultimately interpreted in terms of a five-level tree, ranging from session and modes at the top to operations and actions at the bottom. In the new grammar, some modes are "deeper" than others; that is, they may include submodes that serve strategic purposes, resulting in parse trees whose branches differ in depth. A second distinction is that we have incorporated the computer tool, itself -- used by the users of our systems to mediate their cognitive behavior -- as a fundamental part of the task model. That is, when someone uses a particular tool for a task -- such as our writing environment -- that tool affects the user's thinking -- the processes they use, their strategies, etc. Consequently, we are including the user's attention to the tool -- manifest as a sequence of operations that address the system rather than the conceptual artifact being represented -- as a mode of thought analogous to other modes, such as exploring or organizing.

To gain a feel for the ATN task model, consider Figures 5 & 6. At the highest level, we presume that users are *working*; for whatever reason, they decide to write a document. To do so, they engage a mode that contains a strategic model of the process. Here, they can *explore concepts*, *develop* the *structure* of the document or its *expression*, or they can consciously address the *tool*. Assume that our writers decide to *explore content*: they then engage a submode in which they may *brainstorm*, build *clusters* of ideas, or *build component structures*. If they decide to *brainstorm*, they can represent a concept (*define* it), *revise* an existing concept, or discard (*delete*) an existing concept from further consideration. *Defining* a concept is a basic cognitive process in our model that is done by completing a small sequence of actions that constitute an *operation*. As the final figure shows, this *operation* and the product it produces are subjected to a *test*; if the conditions are satisfied, the transition in this lowest level ATN network is completed.

Other strategic choices would, of course, lead to other behaviors, other protocol sequences, and other paths through the ATN. But this simple example, we hope, will provide an intuitive sense of the model expressed in the ATN grammar. Of course, other models could be expressed in other ATN network structures. Consequently, our strategy in developing a parser is to develop a general tool that can be used with different ATN grammars.

ATN Grammar Tool

To support development of different ATN grammars, we are building an editor with which to define and edit grammars. The ATN-Editor, shown in Figure 7, supports visual display and direct manipulation of ATN graph structures. It also permits the user to define rules on the arcs; we are attempting to provide general evaluative mechanisms so that most rules can be specified by selecting relevant attribute values and conditions from menus of possible choices. However, the editor will also permit ad hoc SmallTalk programs to be written and applied as tests, when required. The editor provides basic error-checking for common errors, such as unreachable states and pushes to undefined nodes. In the future, we plan to add a real-time interpretive capability so that partially-completed ATN grammars can be tested and refined against sample protocol data. At present, the editor exists as a partial prototype; we hope to complete prototype implementation of all basic functions by the end of the year.

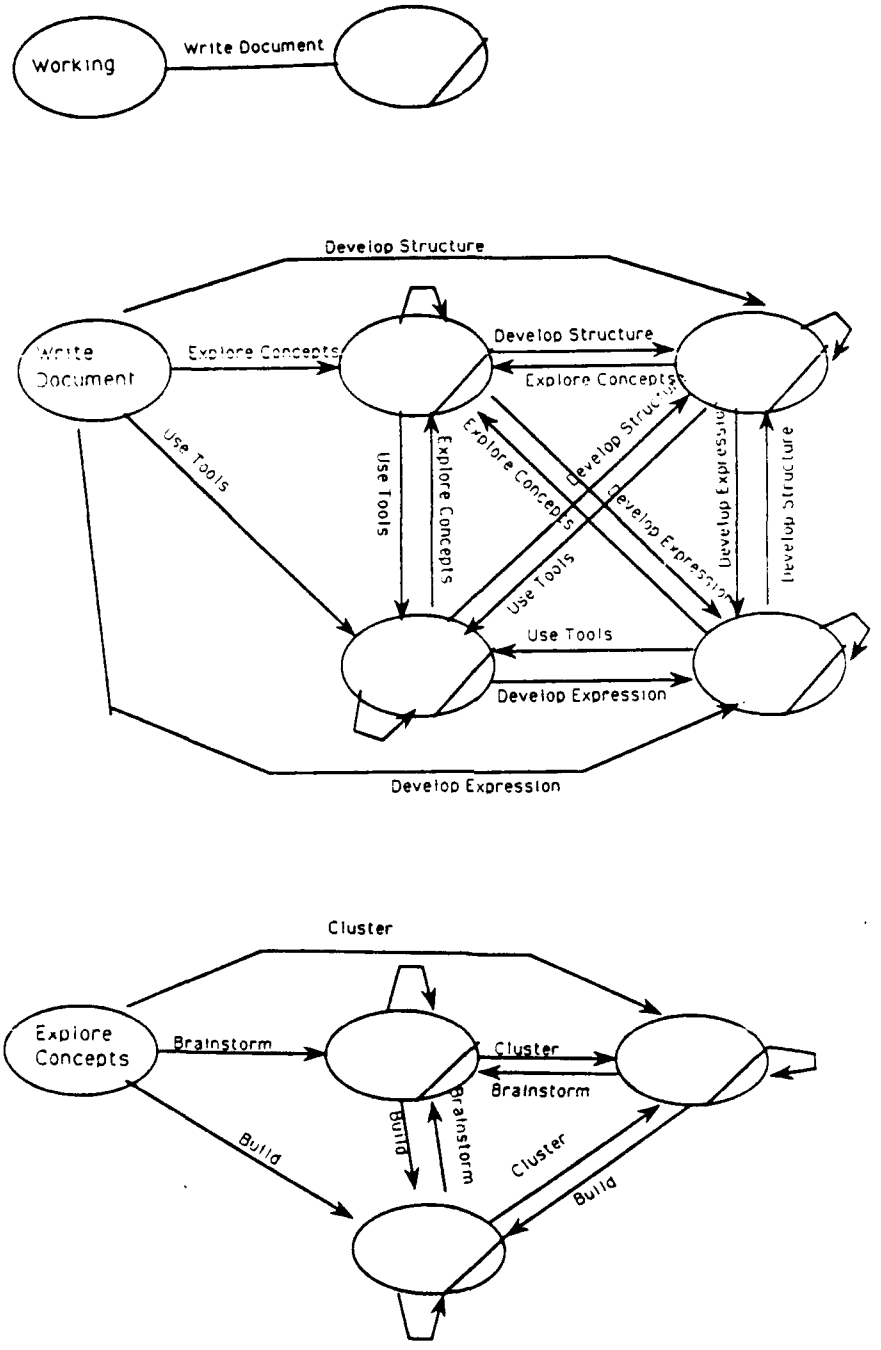


Figure 5:
Model of Computer Mediated Writing Expressed as an ATN Grammar

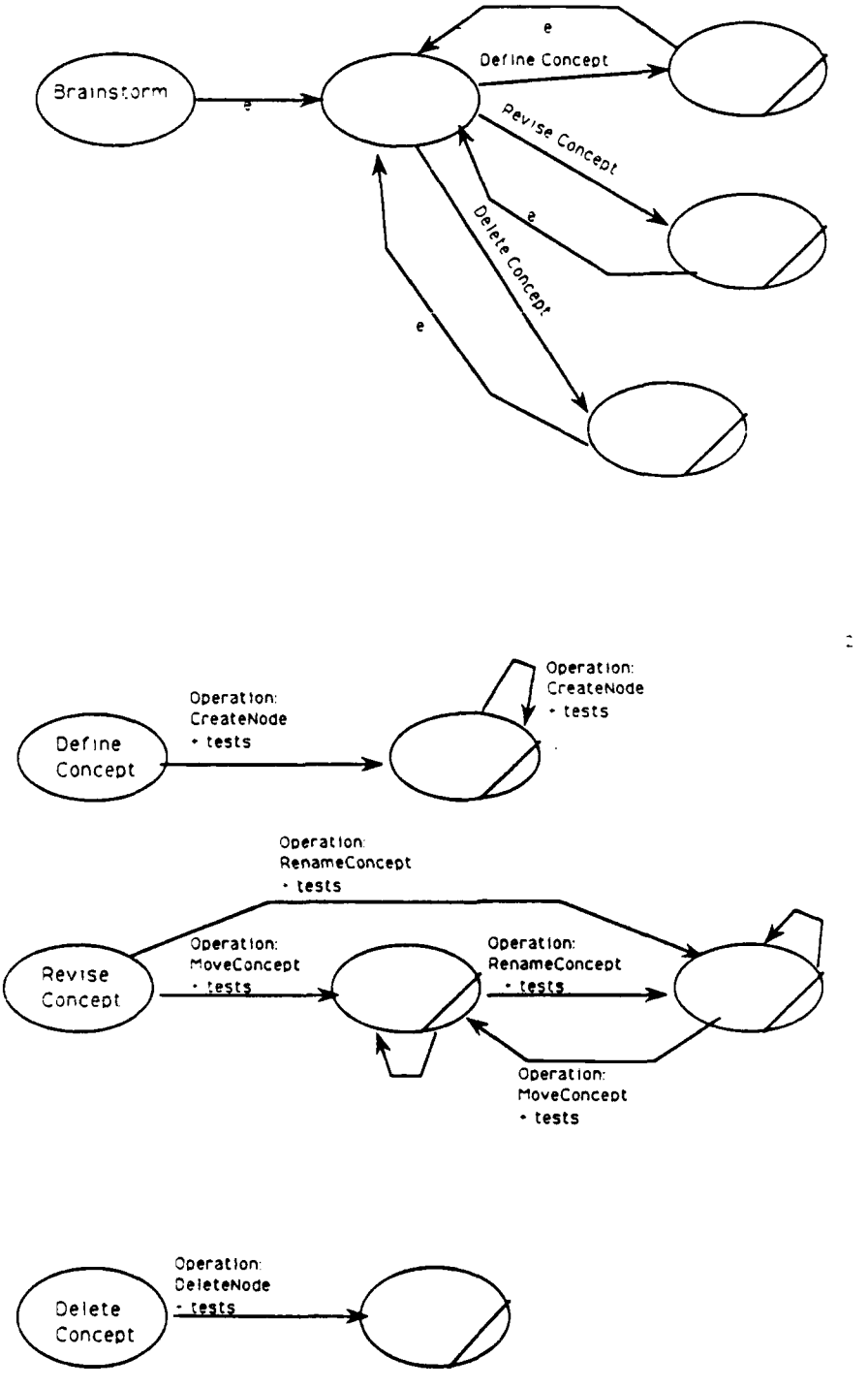


Figure 6:
Continuation of
Model of Computer Mediated Writing
Expressed as an ATN Grammar

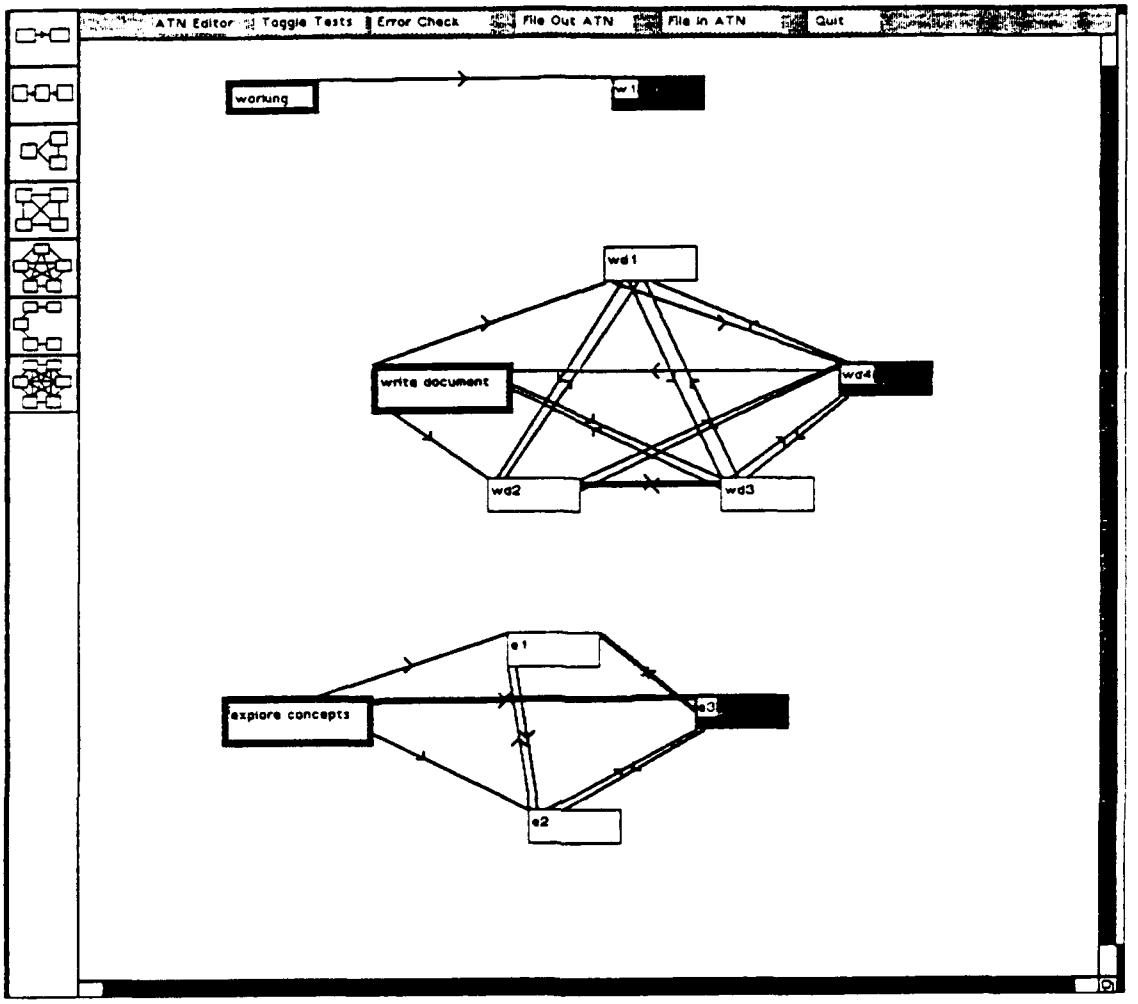


Figure 7:
ATN-Editor

Analytic Tools

Once a protocol for a session has been parsed, the resulting data can be viewed from two perspectives. First, it can be thought of as a parse tree, in which the *session* is the root symbol, consisting of a sequence of *modes*, each comprised of a sequence of cognitive *process* symbols that produce changes to (often intermediate) cognitive *products*; accomplished by sequences of *operations*, each requiring several user *actions*. Thus, each session will produce a parse tree with depth of five levels. While the parse tree can be analyzed directly, we normally focus on a particular level of the parse tree, such as the delta product level or the modal level. Consequently, we can define a second perspective in which a horizontal slice of the parse tree is seen as a sequence of protocol symbols analogous to the original action-level protocol recorded by the system, but at a higher level of abstraction. Each such level defines the user's behavior from a different cognitive perspective -- e.g., as a sequence of shifts in cognitive mode or as a shift from working on one (intermediate) cognitive product to another.

Analysis, of course, is an iterative, open-ended process; but each iteration normally consists of two computer processing steps: data are first filtered to produce one or more measures; second, these measures -- usually aggregated for all subjects taking part in an experiment or in one condition or category -- are then analyzed using a standard statistical package. In the remainder of this section, we describe our work supporting these two steps.

Filters

Filters are independent computer programs that scan a protocol sequence -- at the action level as recorded by the application program or at an intermediate level in the parse tree -- and produce values or sequences of values that are subsequently analyzed statistically or are displayed. The measures they produce quantify some pattern of the user's cognitive behavior from a particular analytic perspective. In a series of experiments concerned with differences in novice and expert writers' strategies, some of the specific issues we are interested in, for which we developed filters/measures, included the following:

- will experts produce more complex hierarchical structures and/or will they develop them more quickly?
- will novices struggle more with the development of the structure for their papers since they must develop an organization for the information from scratch and cannot simply adapt knowledge already in memory?
- will experts write better papers and will they write them more efficiently?
- will experts include in their documents more ideas outside the source materials provided than novices?
- will novices tend to follow a last-item-generated-as-cue retrieval strategy versus a strategy based on conceptual structure, expected of experts?

- will the main topics of experts be more similar across subjects than those of novices?

To address these issues, we have developed the following filters:

Top-Down Index

Using the Writing Environment (Smith, Weiss, & Ferguson, 1987; Smith, Weiss,Ferguson,Bolter, Lansman, & Beard,1987) , writers can develop the ideas for their paper in whatever order they like. One strategy for creating ideas is to create superordinate topics first, followed by subordinate topics. This would be similar to filling out an outline from the top to the bottom. We call such an approach a *top-down strategy*. An alternative method is to generate subordinate topics first, group them, and then create appropriate superordinate topics. This would be a *bottom-up strategy*. The *top-down index* indicates the degree to which writers followed one or the other of these strategies.

To calculate top-downness, we compute the percent of nodes (omitting the root node) that were created after their superordinate node was determined. If writers had followed a purely bottom-up strategy, their top-down scores would be 0.00; if writers followed a purely top-down strategy, their scores would be 1.00. In our experiments, subjects' scores of top-downness ranged from 0.5 to 1.0, with a mean of 0.80.

Stage Index

This index assesses the extent to which planning time precedes writing/revising time. (The name, *Stage Index*, derives from an early, now discredited, theory of writing that asserted that writers progress linearly through three stages -- planning, writing, and revising.) In order to understand the Stage Index, imagine computing for every minute of writing time the *proportion* of total planning time that preceded that minute of writing. These proportions are averaged across all the minutes of a writing session to compute the Stage Index. For example, if a subject had completed all planning before beginning to write, then for each minute of writing the proportion of planning that preceded that minute would have been 1.0 and the average, the Stage Index, would be 1.0. The index can vary between close to 0 and 1.0. (It can't be 0 since subjects using the Writing Environment must create at least one node in either Network or Tree Mode before they can begin to write.)

Struggle Index.

This index assesses the difficulty, or struggle, writers have in developing an organizational structure for their papers. Difficulty in developing a structure might be exhibited by the number of actions in Network and Tree Modes versus the size of the structure produced. The index is defined as follows:

STRUGGLE = PLANNING_OPS / TOTAL_NODES

where:

PLANNING_OPS = # of create node + create link + delete (node, link or tree) + move (node or tree) + rename

TOTAL_NODES= # of nodes in final structure (tree)

Conceptual vs. System Operations Index

Operations in Network and Tree Modes can be classified as either being a *conceptual* operation or a *system* operation. Conceptual operations are operations which directly contribute to the development of the conceptual structure for the writer's document. System operations are those that are concerned with controlling the Writing Environment, *per se*. More precisely,

INDEX = CONCEPT_OPERATIONS / SYSTEM_OPERATIONS

where:

CONCEPT_OPERATIONS = delete (node, tree or link), create node, create link, rename & context (roaming)

SYSTEM_OPERATIONS = system, layout & view

The set of filters is, of course, open-ended. They are dependent on the goals and perspectives of specific studies. Those described above constitute our current inventory, but as we address new research issues, we expect the list to grow.

Statistical Analysis

Data -- consisting of simple counts of protocol symbols at various levels of the parse tree as well as values produced by filters -- are analyzed using a standard statistical package -- currently, SPSSX (see Lansman, Smith, & Weber, 1990 for an example).

These data are written to a file and the file processed by the statistical utility in a conventional manner. We would like to buy or develop a graphics-based data-flow control program to assist with the management of statistical analyses. We can envision a system in which one represents sets of data and statistical functions by icons. One could then perform a specific analysis on a given set of data by simply drawing a link between their corresponding icons and then represent the results by a new data icon. Some of our ideas about such a system are described in Young and Smith [1989], but we have not begun work on this project.

Display Tools

The goal of our research in protocol methodology is to automate the recording and analysis of protocols as much as possible. However, human beings, not machines, must ultimately decide what the data tell us. Thus, our strategy is not to remove human researchers from the process, but rather to provide them with tools to help them manage, analyze, and interpret protocol data. Particularly important are display tools to help researchers visualize the data.

We have built two types of display tools: *static* and *animated*. As the terms imply, static tools provide a snap-shot of numerical or time-series data for one or more user sessions from a particular analytic perspective. By contrast, animation tools provide a dynamic view of a session that unfolds in time, as the session is replayed. Both types of tools are discussed below and illustrated in accompanying figures.

Static Tools

The *Events-Time Distribution Tool*, illustrated in Figure 8, provides a static representation of the frequency with which user events, grouped by analyst-determined categories, are distributed over a session. The display is divided into two panels. The

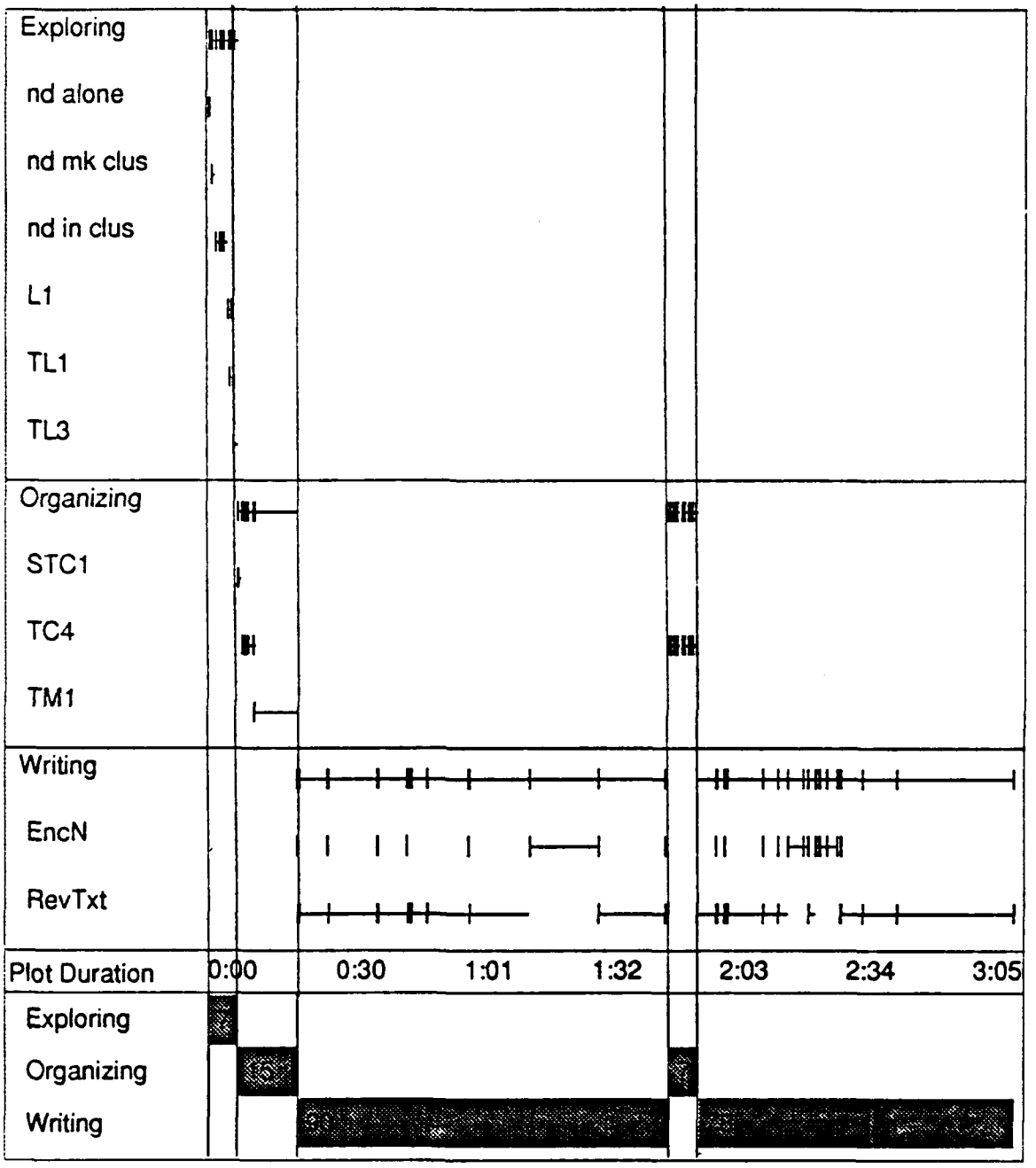


Figure 8:
Events-Time Distribution

left panel lists a taxonomy of events that was previously defined by the user in a tree representation. Frequency distributions for each type of event are displayed in the right panel of the screen, in which session time extends from left to right. The instances of each event type are represented by a series of tick marks (short vertical lines) at the time each event begins. A horizontal line extending from the center of each tick mark (the tail) indicates the duration of the event. The panel is divided into horizontal bands that correspond to the categories of the taxonomy defined in the left panel. A row of tick marks, thus, appears for each type of event and for the cumulative total for all events within a category. Changing the structure of the tree in the left panel changes the organization of the distributions in the right panel. The bottom section of the right panel shows a histogram that can display either the total number of events within each category or the total duration of events within the category. In Figure 8, the events are taken from the delta product level of the grammar, discussed above. They have been organized into three categories - *explore, organize,* and *write.*

To display the behaviors of several subjects for purposes of comparison, we have developed an abbreviated version of the Events-Time Distribution tool. This tool organizes events into only two categories: *planning* -- composed of exploration and organizing events -- and *writing* -- composed of writing, per se, and revision events. Figure 9 shows an example screen of abbreviated distributions for four subjects. The particular classification of events is under the control of the researcher and can be changed to fit his or her research perspective..

Animation Tools

The protocol data we are concerned with is inherently temporal: events take place one after the other in time. Consequently, users' strategies can be described as patterns in behaviors that occur over *time.* To enable researchers to literally see these patterns, we are developing a set of tools in which time functions as the independent variable. We call these time-oriented displays, *Animation Tools.*

Animation Tools actually refers to a collection of display tools from which the researcher may select and arrange particular configurations to suit particular needs. We typically work with a set-up of three coordinated workstations. One workstation runs a replay of the session being viewed. The displays on the other two workstations include various data representation windows plus control and configuration windows. The various windows can be arranged to suit the researcher's preferences. Figure 10 shows a typical configuration for three workstations; we will refer to it in the discussion that follows.

The left screen in the configuration is the replay display. It is identical to the replay tool discussed above. Here, it serves as the central clock that coordinates all of the associated displays. As the events unfold in time on the replay screen, the other display tools -- shown on the other screens -- update their displays accordingly.

The animated display is controlled by the researcher from the middle screen, shown in context in Figure 10 and enlarged in Figure 11. The largest window is the configuration window, in which the researcher selects the particular protocol for display and the various display tools included in the configuration. Above that is the control window used to stop and start the replay and, hence, the animation. So long as the cursor is in this window, the replay will continue and the other windows will update their displays accordingly; however, the researcher may stop the replay by moving the cursor to any of the other windows. Two comment windows are shown on the right side of the screen. They are text editors in which written comments or other data can be noted and linked to a particular time or event in the protocol. Thus, the researcher may stop the animated display, by moving the cursor from the control window to one of the

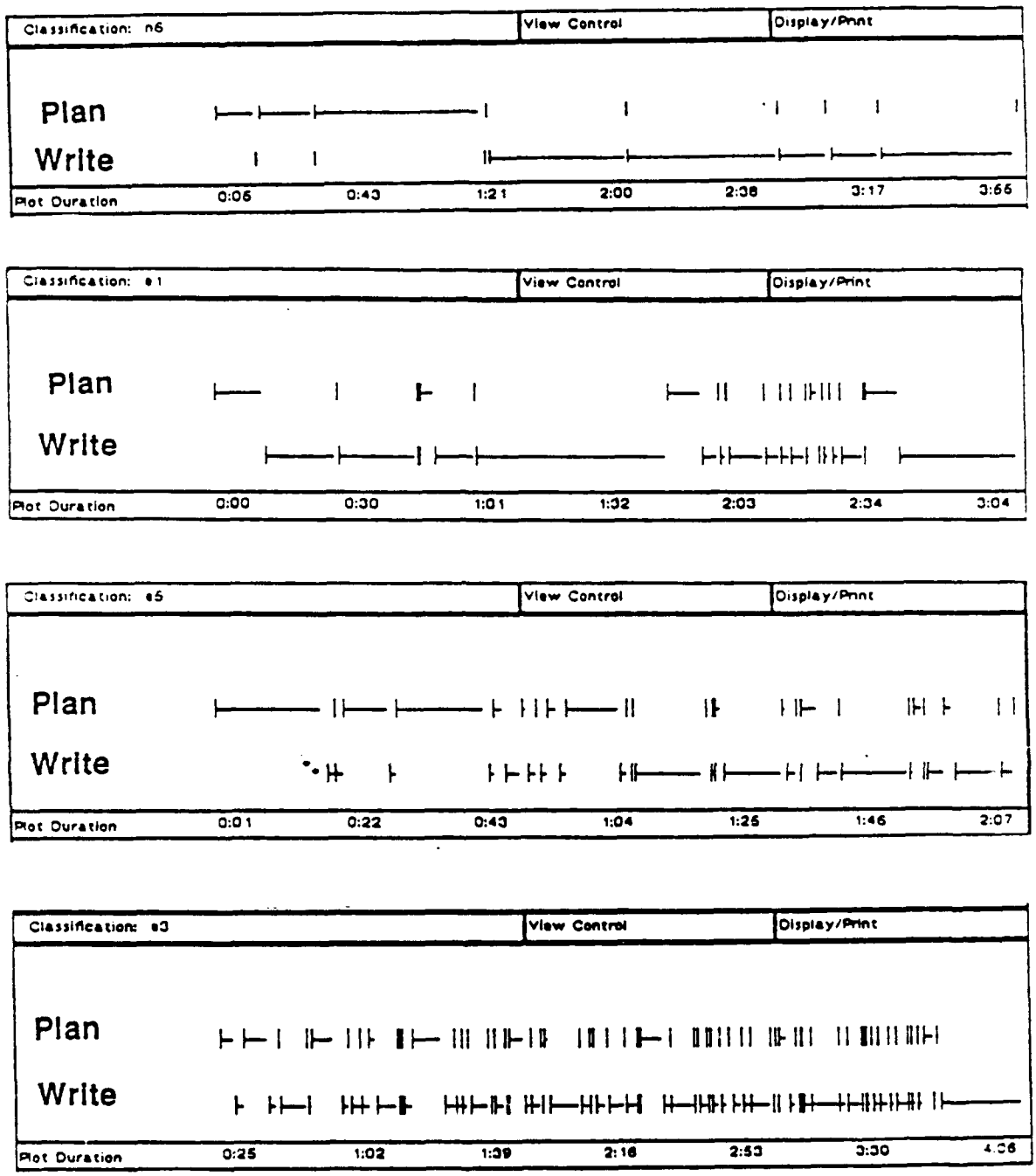


Figure 9:

Abbreviated Events-Time Distribution

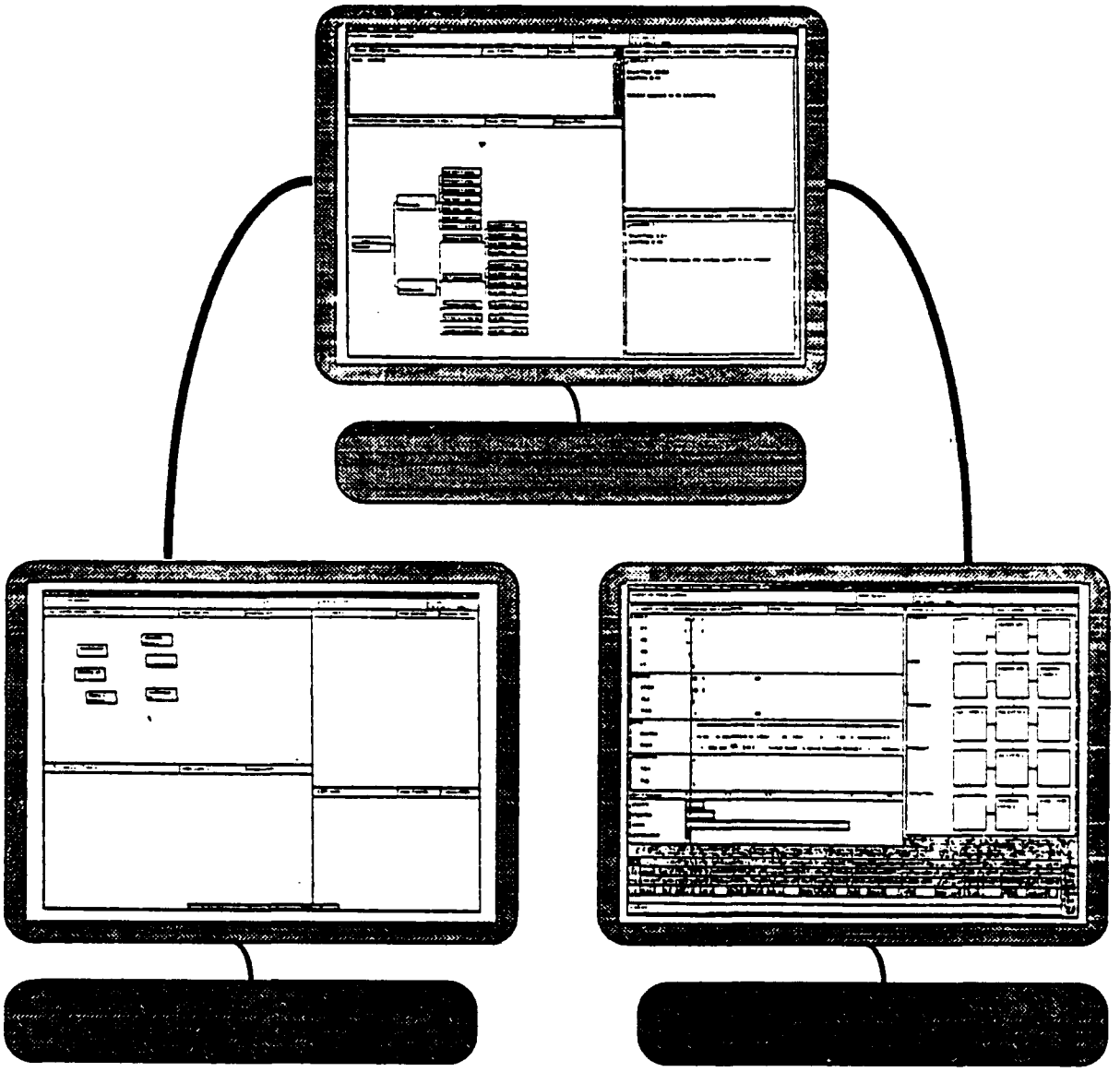


Figure 10:

Animation Display -- Overview

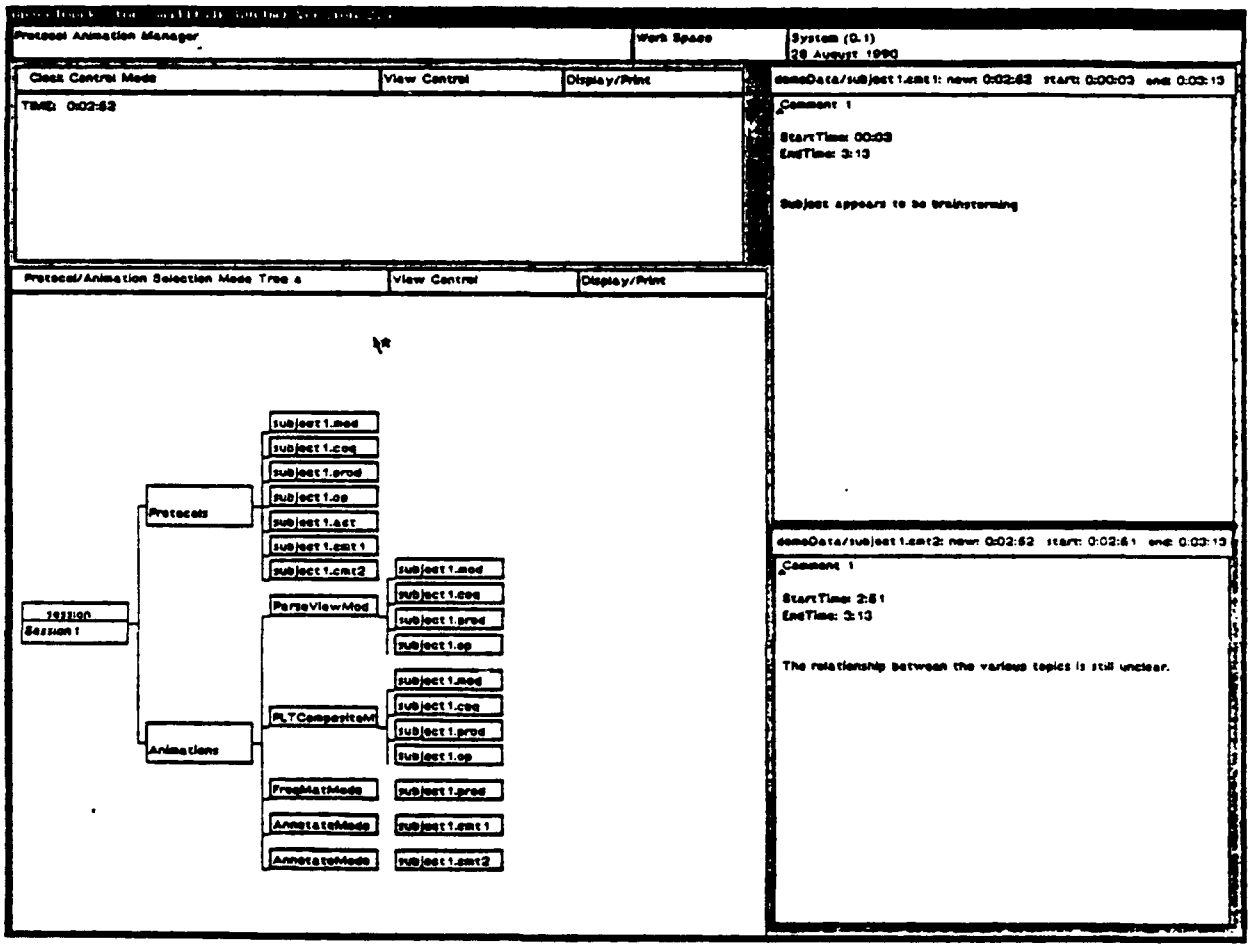


Figure 11:

Animation Display

Configuration, Control, & Comment Windows

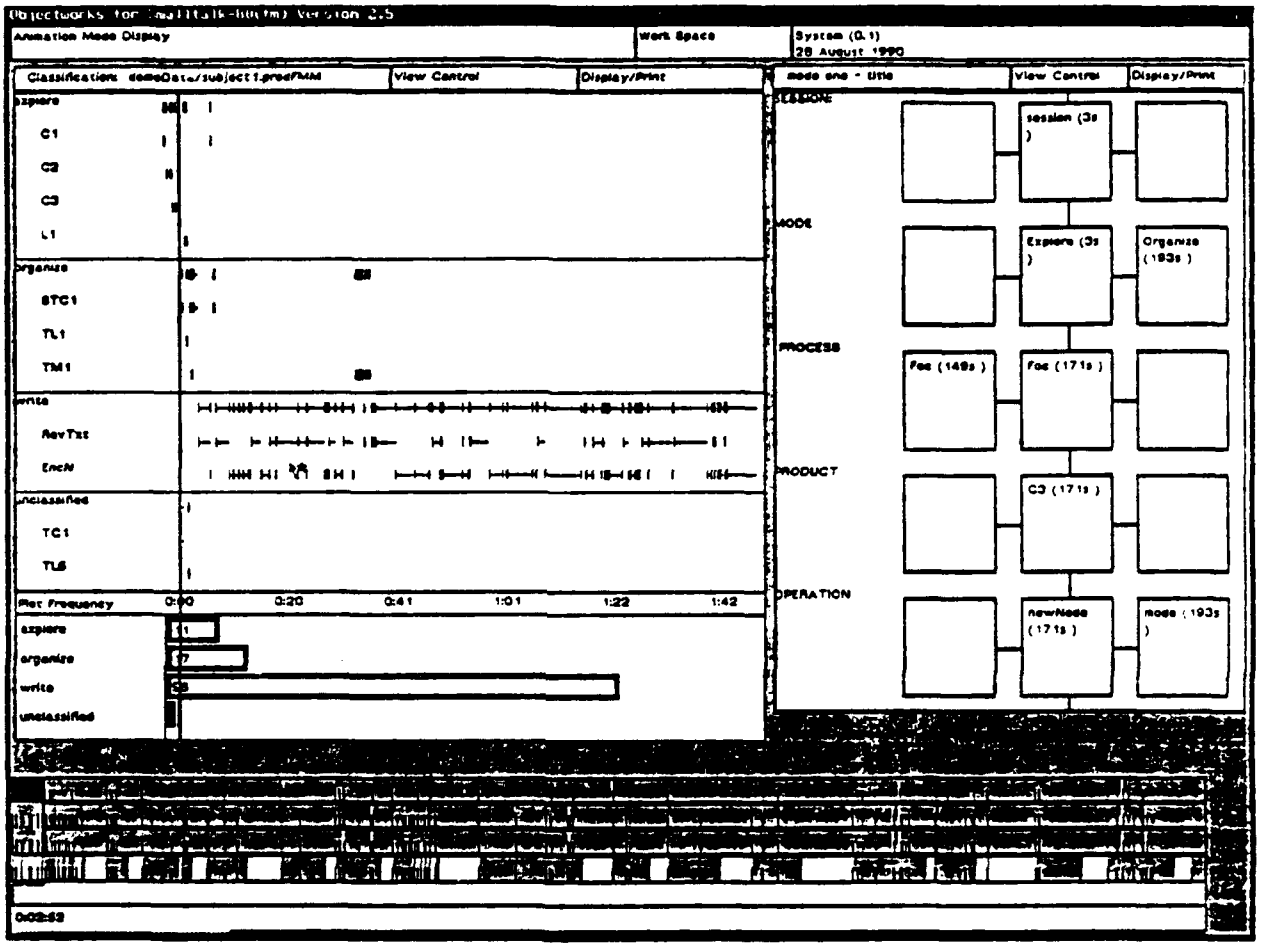


Figure 12:

Animation Display

Dynamic Events-Time & Parse Tree Windows

comment windows, and then write an observation about what was just seen in the session. When an annotation window is opened, it is stamped as being opened at time T1; when the researcher no longer wants the comment to be shown, he or she closes the window and it is stamped as closing at time T2. When the session is replayed at some future time, the comment will appear and be displayed from time T1 until time T2. The researcher may set-up any number of comment windows that can be used to record different kinds of information. Thus, for example, a researcher might have three separate comment windows in which a transcription of a subject's concurrent think-aloud protocol is recorded in one, a cued retrospective protocol stimulated by the subject's viewing a replay of his or her session in a second, and the researchers own observations in the third. In the future, we will experiment with linking video and voice displays to the time-line for the session and coordinating display of this information with the replay.

The screen on the right in figure 10 and enlarged in Figure 12 contains two additional animated displays. On the left is a dynamic version of the Events-Time Distribution tool described above. It is similar to the static tool except that as the session unfolds, the vertical line (shown near the left edge in Figure 12) slowly moves across the display to the right to indicate the current time and event in the replay. To the right of the events-time distribution is a second window, showing a vertical slice of the parse tree produced by the cognitive grammar described above. It shows four levels of the parse tree corresponding to the *cognitive mode, process, cognitive product,* and *operation* levels of the grammar. In the center column is the grammatical interpretation for the current event. The display also shows the preceding and succeeding symbols at each level of the parse tree. As the replay runs, the information within these boxes changes. The Parse Tree Display tool enables the researcher to compare the interpretation produced by the grammar for a segment of a session, as recorded in the parse tree, with the actual events, as shown in the replay. This capability can be used for a variety of analytic purposes as well as to validate and refine a particular grammar.

As noted above, specific display tools can be selected and combined in various ways. Thus, they form an open set; over the next few years, we plan to develop new displays that can be included within animation configurations.

Data Management

Machine-recorded protocols permit researchers to gather large volumes of data. This can be a curse as well as a blessing; anecdotes abound of researchers with, literally, drawers full of keystroke data that have never been analyzed. However, automated tools, such as the grammar described above, can assist analysis and interpretation and, thus, help researchers make more effective use of these data. Nevertheless, problems remain. For studies of differences among groups of subjects, researchers must keep up with particular sets of protocols for particular subjects collected under particular conditions. For longitudinal studies, protocols must be selected by subject and sequenced according to time of generation. During analysis, the researcher must keep track of which data has been processed by which filters and programs. And for animated display, consistent sets of data must be assembled and passed to the system through the appropriate file. Thus, while these problems are an issue for any human-computer interaction study, automated protocol tools magnify the need for effective *protocol data management* tools.

While we recognize the issue, our efforts in protocol data management are still tentative. To date, we have carried out two projects. The first is development of a prototype system for sorting, selecting, and sequencing protocols. The second is a mathematical analysis of these tasks that could provide a formal basis for a

comprehensive protocol analysis environment we hope to build in the future. Our work on these two topics is described in more detail in [Hawkes, 1991].

As discussed above, when one of our application systems, such as the Writing Environment, is being used in a study, the shell in which the program operates is configured so that individual users' protocols are stored within a particular directory of the UNIX file system. Each such protocol includes, along with the transcript of the user's actions, a header that includes information that identifies the particular user, the time and date when the protocol was collected, and other similar data. Soon after the recording, the experimental monitor adds additional information to each protocol header, such as whether the user is an "expert" or "novice" with respect to the task, the judges' evaluation of the document produced, etc. These data are stored in the header in the form of *attributes* that identify the type of information -- e.g., date -- and *values* that record particular data -- May 19, 1991. When an experiment is complete, all of the protocol transcripts for that experiment are stored in the same directory under identifying file names

We have developed a prototype tool to help the researcher sort, select, and sequence protocols according to particular attributes and values stored in their respective headers. The tool lets the researcher work with one directory of protocols at a time. The researcher sorts and/or selects from the set in that directory by successively designating attributes. Protocols are then ordered or selected according to the values in their headers for the specified attribute. Each designation of an attribute produces a further ordering (for a sort) or reduction in the number (for a select) in the set of protocols. The final ordered/selected set of protocols is represented as a tree, as shown in Figure 13.

To understand how the tool works, imagine that all of the protocols stored in a single directory are first represented as a "bushy tree" in which the directory, or the experiment, corresponds to the root node of the tree and each individual protocol corresponds to a child node of the parent/experiment. Thus, the tree starts with a depth of one.

At the bottom of the screen shown in Figure 13, aligned under the column(s) of child/protocol nodes, is a row of "buttons" marked *KEY:1 . . . Key:11*. When the user selects KEY:1, a menu is shown that lists each attribute included in the protocol header for that experiment, such as *subject, date, experience,* etc. The user selects one of the attributes on the menu, such as *experience*; the system then prompts for *sort* or *select*. If *select* is designated, the user is prompted for a value, such as *expert*. The system will then select only those protocols that include in their headers the attribute, *experience*, whose value is *expert*; all others -- i.e., those for *novices* -- will be ignored for the duration. If, instead of *select*, *sort* is designated, all of the protocols in the set will be reordered according to their value for the designated attribute. Thus, by successively designating select or sort for different attributes, the researcher can select and order subsets of protocols for particular analytic purposes.

In the example shown in Figure 13, the particular set of protocols is identified as *PTC*. From this collection, the researchers has selected protocols for those users classified as *experts* and, secondly, those produced on *4 August, 1988*. Having narrowed the collection of protocols to that subset, the researchers has then sorted them according to two sort keys, applied in sequence. The protocols are sorted, first, on *user*, then on scores assigned by human judges -- called *grade*. This procedure produces the final sequence of protocols, shown as the ordered leaves of the tree. That sequence can be changed by going back to an earlier choice/column and designating a different attribute or value for that column. Once the desired sequence protocols has been identified, they can then be further analyzed or displayed, as needed.

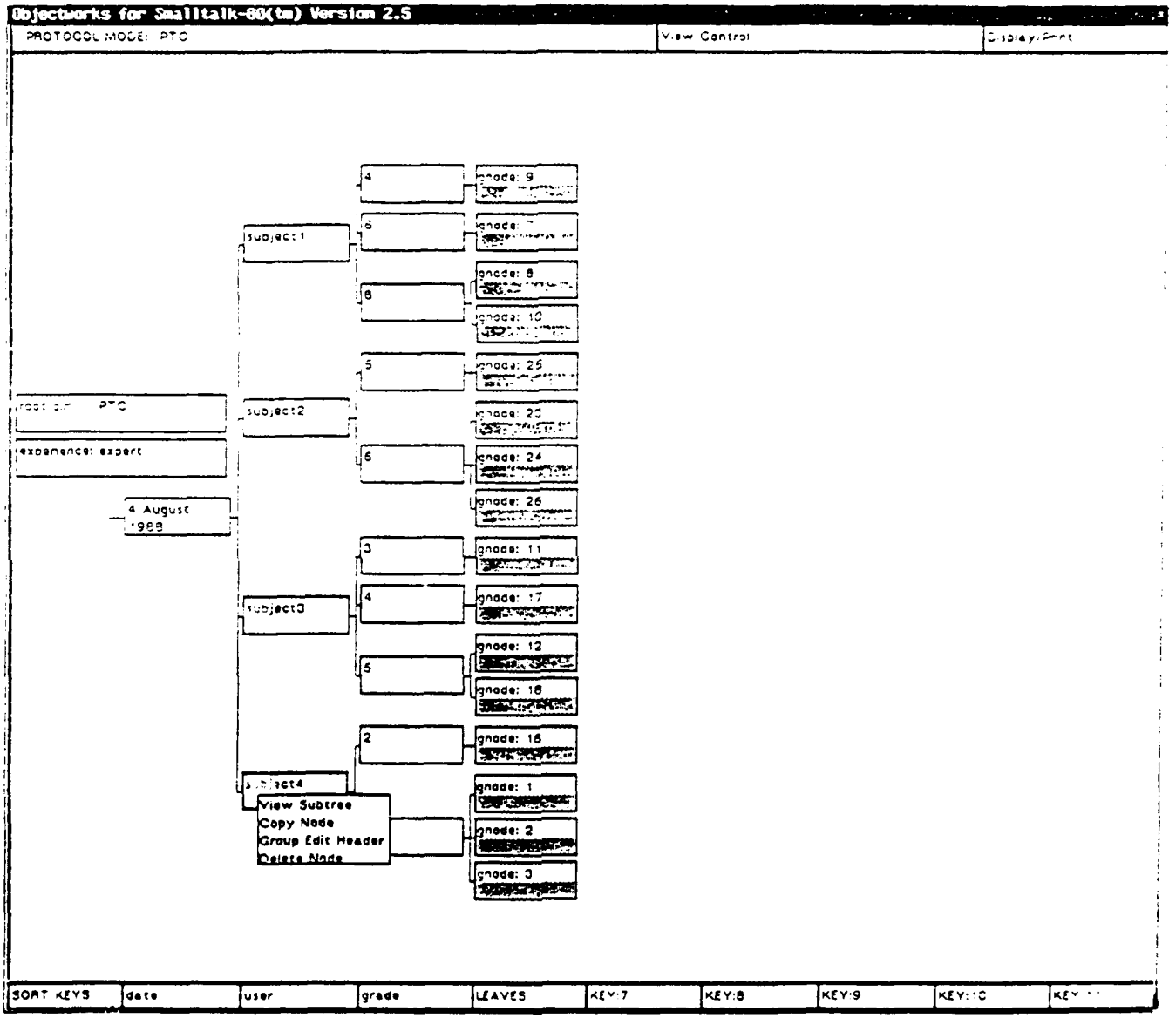


Figure 13:

Protocols Sort/Select Management Tool

The protocol selection tool described above is a prototype. We are in the early stages of designing a comprehensive system to manage all stages of the analysis and to help researchers work with large numbers of protocols for both individual users as well as groups working collaboratively. To provide a well-defined basis for that system, we have analyzed in detail the formal characteristics of trees of objects ordered or selected according to attributes and values, as described above. We have also considered what it means to apply an operator or analytic function to an object, such as a protocol, and under what conditions operators with particular characteristics can be applied to protocols or other data with particular characteristics. The result of this theoretical work is a formal description of sort/select trees of protocol data and associated analytic operators, described in [Hawkes, 1991].

For example, after selecting/ordering a set of protocols, as described above, the researcher might apply a filter program to count the number of planning actions in each protocol and output that number for each protocol. Those numbers could then be divided into two sets and analyzed using an analysis of variance program to see if one group does significantly more planning than the other. However, before doing the analysis, we would like to be sure that the data are suitable for an analysis of variance. For example, a second filter might produce a distribution of numbers for each protocol, rather than individual values, by counting and outputing the number of planning actions found per fixed number of actions or fixed duration of time within a protocol. These data, however, would not be suitable for analysis of variance, but they might be properly analyzed using other functions to see whether one group does more early planning , versus later planning, in the task than the other group.

The general case, then, is a formal description of *attribute-value trees* and *attribute-value operators* and the conditions under which an operator with particular characteristics can be applied to a particular ordered set of protocols. While our mathematical description is oriented toward protocol data, much of the formalism is general and could be applied to other data with specified format types and semantics and an analogous set of operators. Thus, for example, we can imagine a general-purpose direct manipulation interface for a statistical system that would provide graphic icons for statistical functions and data sets. One could then construct data flow diagrams to control and/or represent a multi-stage analysis. By incorporating into the system rules such as those described above, the system could insure that any given analysis will be meaningful at least with respect to the organization of the data and their semantics and a given statistical function. We have sketched such a design in Young and Smith [1989].

Conclusion and Future Work

Summary

We have described our work in protocol tools and methods done over the past seven years. More specifically, we have discussed the notion of viewing as a *language* sequences of machine-recorded protocol symbols representing users' actions while they work with a particular computer system. We pointed out reasons for recording these data at the level of *action*, rather than keystroke, and we pointed out some of the strengths and limitations of this form of data in relation to other forms of protocols, such as think-aloud reports and video/audio records.

One implication of viewing machine-recorded protocols as a language is that it leads to the idea of modeling and analyzing/parsing these data with *cognitive grammars*. Since such a grammar maps, in a context-sensitive way, users' actions into symbols that represent (intermediate) cognitive products, processes, and modes, it can be viewed as a formal cognitive model of users behaviors for a given task mediated by a given

computer system. This is a powerful concept, and we can foresee a variety of uses for such grammars in addition to studying human-computer interaction. For example, building and refining a grammar could be linked with building and refining the computer system, itself, leading to a methodology for developing *theory-based systems.* Grammars running concurrently while a user works with a system could support a variety of intelligent, real-time functions, such as invoking context-sensitive help and intelligent tutoring systems as well as launching background support functions at times appropriate to users' current strategies or task conditions. But to be effective, these grammars must be accurate; we are currently working on strategies and tools to help researchers refine their grammatical models by comparing grammar-produced interpretations with users' verbally produced accounts and/or the conceptual products they produce.

We then described a series of special-purpose programs we have developed that filter the parsed protocols and extract various quantitative measures. These data, in turn, are analyzed using a standard statistical package.

Next, we described a set of display programs we have built to help researchers understand and interpret these data. These tools take two forms: *static* and *animated.* Static tools represent a snapshot of the data for a session of group of sessions. Animated tools are controlled by the replay function and show the unfolding of an analysis or interpretation relative to the session as a whole.

Finally, we discussed some of our preliminary work concerned with *managing protocol data*, including their analyses and display. Machine-recorded protocols combined with automated tools for their analysis could change the nature of human-computer studies, making practical extensive longitudinal studies, studies of naturalistic behaviors, and studies that include large numbers of subjects. At the same time, automated techniques also raise significant problems in data management. One could develop a separate data management system; however, viewing data management as the core activity leads to the notion of a comprehensive system for managing all aspects of protocol analysis -- from initial collection through analysis and display, including development and refinement of grammars. We have begun designing such a system, which we hope to build over the next few years.

Methodological and Theoretical Issues

Tools and techniques such as these have several methodological and theoretical implications. A very rich source of insight into complex mental processes have been the verbal reports of subjects prompted by a researcher to think-aloud as they carry out a task. However, these data raise significant theoretical as well as practical issues, including their reliability, completeness, and possibly distorting effects on task behavior. As we noted above, these effects are expected to be most significant for tasks that are visual and abstract -- the very conditions created in current graphics-based direct-manipulation computer systems. Consequently, we badly need a clearer understanding of the relative strengths and weaknesses of think-aloud, cued retrospective think-aloud, and machine-recorded protocols. Since the concerns raised for think-aloud protocols raise basic epistemic issues, designing experimental studies to probe the effects and validity of thinking-aloud has been problematic. However, comparing verbal reports with machine-recorded protocols offers an approach in which these issues can be addressed *directly*. In the near future, we plan to carry out a study in which we will have subjects use our system both while thinking-aloud and not thinking-aloud. We will then compare the two forms of protocols for completeness and to see if users' behaviors are different when they narrate their thinking.

A second issue concerns the task models we have built in the form of grammars. These models are defined in terms of *modes* -- consisting of interdependent combinations

of *goals, products, processes* and *constraints.* In the future, we will attempt to refine these models in order to make them more accurate and more sensitive; that is, to be sure we have identified a precise and sufficient set of modes for a given task, comprised of the right set of components. We will also try to develop more sophisticated rules for interpreting and predicting shifts from one mode to another, that take into account user's overall strategies and specific conditions in the current mode.

We will also develop additional modal models for other tasks -- specifically, software development. Writing and software development share many of the same modes, particularly during planning; during other phases, they have different, but corresponding modes -- for example, writing, per se, and coding play similar roles. Consequently, skills developed for one task could transfer to other tasks, once individuals perceive the relationship. We will probe this issue in a series of dual-task experiments.

This line of research also leads to a more general notion of a *cognitive architecture* -- expressed in terms of modes and strategies -- for describing the large class of tasks that involve building large, complex structures of ideas. A long-term goal will be to clarify this architecture by developing and comparing specific modal models for different tasks.

Group Protocols

The work described above is concerned with the mental behavior of individuals. We are currently extending our system-building research from computer tools to support individual users to tools to support groups of users working collaboratively. A key goal for collaborative work is the development of conceptual artifacts that have the same consistency and integrity as those produced by individual thinkers under ideal conditions. However, some groups are much more successful at doing this than others. To study and to understand this more complex form of intellectual behavior will require an analogous set of protocol tools for group support systems. It will also require a more complex cognitive architecture -- one that characterizes the cognitive and social interactions of groups of individuals engaged in a more extensive set of *modes of activity.* Eventually, we hope that this line of research will lead to a notion of *collective cognition.* While we can partially envision what these concepts will look like, defining them with sufficient precision for them to be truly useful poses a formidable challenge.

Acknowledgments

This research was supported by the National Science Foundation (Grants # IRI-8519517 and # IRI-8817305), The Army Research Institute (Contract # MDA903-86-C-345), and The Office of Naval Research (Contract # N00014-86-K-00680). A number of individuals have contributed both ideas and effort to the work described here; they include: Marcy Lansman, Steve Weiss, Dick Hayes, Gordon Ferguson, Oliver Steele, Mark Rooks, Doug Shackelford, Rick Hawkes, Matt Barkley, Irene Weber, Hong Li, and John Walker.

References

Card, S.K., Moran, T.P., & Newell, A. (1983). *The Psychology of human-computer interaction.* Hillsdale, NJ: Lawrence Erlbaum Assoc.

Claparède, E. (1934). Genèse de l'hypothèses. *Archives de Psychologie, 34,* 1-155.

Ericsson, K.A.; & Simon, H.A. (1984). *Protocol analysis: Verbal reports as data.* Cambridge, MA : The MIT Press.

Ericsson, K.A.; & Simon, H.A. (1980). Verbal reports as data. *Psychological Review, 87,* 215-251.

Fisher, C. (1988). Advancing the study of programming with computer-aided protocol analysis. In *Empirical Studies of Programmers.* (Eds.) G. Olson, E. Soloway, & S. Sheppard. Norwood, NJ: Ablex Publishing Corp.

Flower, L.S. & Hayes, J.R. (1984). Images, plans, and prose: The representation of meaning in writing. *Written Communication, 1,* 120-160.

Foley, J. D.; & Wallace, V. L. (1974). The art of natural graphic man-machine conversation. *Proceedings of the IEEE,* 62, pp. 462-471.

Hawkes, R.M. (1991). *Mathematical basis for a system to manage automated protocol analysis.* Chapel Hill, NC: UNC Department of Computer Science Technical Report # TR91-024.

Hayes, J.R., & Flower, L.S. (1980). Identifying the organization of writing processes. In L. Gregg & E.R. Steinberg (Eds.), *Cognitive Processes in Writing* . Hillsdale, NJ: Erlbaum, 3-30.

Henry, L.K. (1934). The role of insight in the analytic thinking of adolescents. *Studies in Education, University of Iowa Studies, 9,* 65-102.

Kieras, D.; & Polson, P. (1985). An approach to the formal analysis of user complexity. *International Journal of Man-Machine Studies,* 22, 365-394.

Lansman, M., Smith, J.B. & Weber, I. (1990). *Using Computer-generated protocols to study writer's planning strategies.* Chapel Hill, NC: UNC Department of Computer Science Technical Report # TR90-033.

Losada, M.; Sanchez, P.; & Noble, E. E. (1990). Collaborative technology and group process feedback: Their impact on interactive sequences in meetings. In *Proceedings of CSCW'90,* 53-64.

Lueke, E.; Pagery, P.D.; & Brown, C.R. (1987). User requirement gathering through verbal protocol analysis. In *Cognitive Engineering in the Design of Human-Computer Interaction and Expert System,* (Eds.) G. Salvendy. Amsterdam: Elsevier Science Publishers.

Mackay, W. E. (1989). EVA: An experimental video annotator for symbolic analysis of video data. *SIGCHI Bulletin*, 21(2), 68-71.

Newell, A., & Simon, H.A. (1973). *Human Problem Solving.* Englewood Cliffs, NJ: Prentice-Hall.

Nisbett, R.E., & Wison, T.D. (1977). Telling more than we can know: Verbal reports on mental processes. *Psychological Review, 84,* 231-259.

Olson, G. M.; & Olson, J. S. (1991). User-centered design of collaboration technology. *Journal of Organizational Computing*, 1(1), 61-83.

Reisner, P. (1981). Formal grammars and human factors design of an interactive graphics system. *IEEE Transactions on Software Engineering*, 7(2), 229-240.

Sanderson, P. M.; James, J. M.; & Seidler, K. S. (1989). *SHAPA: An interactive software environment for protocol analysis.* Urbana-Champaign, IL: Engineering Psychology Research Laboratory Technical Report # 89-09.

Smith, J. B. & Lansman, M. (1989). A cognitive basis for a computer writing environment. In Britton, B. K. & Glynn, S. M. (Eds.) *Computer writing aids: theory, research, and practice,* Also published as UNC Department of Computer Science Technical Report # TR87-032.

Smith. J. B., Rooks, M.C., & Ferguson, G. J., (1989). *A cognitive grammar for writing: Version 1.0.* Chapel Hill, NC: UNC Department of Computer Science Technical Report # TR89-011.

Smith. J. B., Weiss, S. F., & Ferguson, G. J., (1987). A Hypertext writing environment and its cognitive basis, *Hypertext '87 Proceedings.* New York: ACM Press, pp 195-214. Also published as UNC Department of Computer Science Technical Report # TR87-033.

Smith. J. B., Weiss, S. F., Ferguson, G. J., Bolter, J.D., Lansman, M. & Beard, D.V. (1987). WE: A writing environment for professionals. *Proceedings of the National Computer Conference '87.* Reston, VA: AFIPS Press, 725-736.

Swarts, H., Flower, L.S., & Hayes, J.R. (1984). Designing protocol studies of the writing process: An introduction. In R. Beach & L.S. Bridwell (Eds.), *New directions in composition research.* New York: Guilford.

Trigg, R. H. (1989). Computer support for transcribing recorded activity. *SIGCHI Bulletin*, 21(2), 72-74.

Walker, J. Q., II (1991). Automated analysis of computer-generated software usage protocols: An exploratory study. Chapel Hill, NC: UNC Department of Computer Science, unpublished Ph. D. dissertation.

Waterman, D. A.; & Newell, A. (1971). Protocol Analysis as a task for artificial intelligence. *Artificial Intelligence*, 2, 285-318.

Waterman, D. A.; & Newell, A. (1973). *PAS-II: An interactive task-free version of an automatic protocol analysis system.* Pittsburgh, PA: Carnegie-Mellon University Department of Computer Science Technical Report.

Woods, W.A. (1970). Transition network grammars for natural language analysis. *Communications ACM, 13,* 591-602.

Young, F.W. & Smith, J.B. (1989). *Structured data analysis: A Cognition-based design for data analysis software.* Chapel Hill, NC: UNC Department of Computer Science Technical Report # TR89-027.